



ゲスト シェル

- [Guest Shell について](#) (1 ページ)
- [注意事項と制約事項](#) (2 ページ)
- [Guest Shell へのアクセス](#) (7 ページ)
- [ゲスト シェルに使用されるリソース](#) (7 ページ)
- [ゲスト シェルの機能](#) (8 ページ)
- [仮想サービスのセキュリティ ポスチャ](#) (16 ページ)
- [ゲスト ファイル システムのアクセス制限](#) (18 ページ)
- [ゲスト シェルの管理](#) (19 ページ)
- [仮想サービスと Guest Shell 情報の検証](#) (31 ページ)
- [ゲスト シェルからのアプリケーションの永続的な起動](#) (32 ページ)
- [Guest Shell からアプリケーションを永続的に起動する手順](#) (33 ページ)
- [ゲスト シェルでのサンプル アプリケーション](#) (33 ページ)

Guest Shell について

基盤となる Linux 環境での NX-OS CLI および Bash アクセスに加えて、スイッチは、「ゲスト シェル」と呼ばれる Linux コンテナ (LXC) 内で実行される分離された実行スペースへのアクセスをサポートします。

ゲスト シェル内から、`network-admin` には次の機能があります。

- Linux ネットワーク インターフェイスを介したネットワークへのアクセス。
- スイッチのブートフラッシュへのアクセス。
- スイッチの揮発性 `tmpfs` へのアクセス。
- スイッチの CLI へのアクセス。
- スイッチのホスト ファイル システムへのアクセス。
- Cisco NX-API REST へのアクセス。
- Python スクリプトをインストールして実行する機能。

- 32 ビットおよび 64 ビットの Linux アプリケーションをインストールして実行する機能。

コンテナ技術によって実行空間を切り離すことで、他の Linux コンテナで実行されているホストシステムやアプリケーションに影響を与えずに、アプリケーションのニーズに合わせて Linux 環境をカスタマイズすることができます。

NX-OS デバイスでは、Linux Containers は `virtual-service` コマンドでインストールと管理されます。Guest Shell は、`virtual-service show` コマンドの出力に表示されます。



- (注) デフォルトでは、ゲストシェルは、有効にすると約 5 MB の RAM と 200 MB のブートフラッシュを占有します。Cisco NX-OS リリース 7.0 (3) I2 (1) 以降、Guest Shell は約 35 MB の RAM を占有します。Guest Shell が使用されていない場合は、`guestshell destroy` コマンドを使用してリソースを再利用します。



- (注) Cisco NX-OS 7.0(3)F3(1)以降、Guest Shell は Cisco Nexus 9508 スイッチでサポートされます。

注意事項と制約事項

Guest Shell には、次の注意事項と制限事項があります。

すべてのリリースに共通の注意事項



- 重要** Guest Shell のインストール内でカスタム作業を実行した場合は、アップグレードを実行する前に、ブートフラッシュ、オフボックスストレージ、またはゲストシェルルートファイルシステムの外部に変更を保存します。

`guestshell upgrade` コマンドは、本質的に、`guestshell destroy`と`guestshell enable`を連続して実行します。

- `run guestshell` CLI コマンドを使用して、Cisco Nexus デバイスのゲストシェルにアクセスします。`run guestshell` コマンドは、ゲストシェルにアクセスするために使用される `run bash` コマンドに相当します。このコマンドを使用すると、Guest Shell にアクセスして `bash` プロンプトを取得したり、ゲストシェルのコンテキスト内でコマンドを実行したりできます。このコマンドは、パスワードなしの SSH を使用して、デフォルトのネットワーク名前空間にある `localhost` の使用可能なポートに接続します。
- `sshd` ユーティリティは、`localhost` でリススンして、ネットワーク外からの接続試行を回避することにより、Guest Shell への事前構成された SSH アクセスを保護できます。`sshd` には次の機能があります。

- これは、パスワードにフォールバックしないキーベースの認証用に構成されています。
- Guest Shell の再起動後にゲストシェルにアクセスするために使用されるキーを読み取ることができるのは root だけです。
- root のみがホスト上のキーを含むファイルを読み取ることができ、ホスト bash アクセスを持つ非特権ユーザーがキーを使用して Guest Shell に接続できないようにします。ネットワーク管理者ユーザーは、Guest Shell で sshd の別のインスタンスを開始して、Guest Shell に直接リモートアクセスできるようにすることができますが、Guest Shell にログインするすべてのユーザーには、ネットワーク管理者権限も与えられません。



(注) ゲストシェル 2.2 (0.2) で導入されたキーファイルは、ユーザーアカウントが作成されたユーザーに対して読み取り可能です。

さらに、Guest Shell アカウントは自動的に削除されないため、不要になったときにネットワーク管理者が削除する必要があります。

Cisco Nexus リリース 7.0(3)I5(2) を使用した 2.2 (0.2) より前の Guest Shell インストールでは、個々のユーザーアカウントが動的に作成されません。

- すぐに使用できる新しい Cisco Nexus スイッチに Cisco Nexus シリーズスイッチソフトウェア リリースをインストールすると、Guest Shell が自動的に有効になります。その後の Cisco Nexus シリーズスイッチソフトウェアへのアップグレードでは、Guest Shell は自動的にアップグレードされません。
- Guest Shell のリリースでは、配布または配布のバージョンが変更されると、メジャー番号が増分されます。
- Guest Shell のリリースでは、CVE が解決されるとマイナー番号が増分されます。Guest Shell は、CentOS が公開した場合にのみ CVE を更新します。
- **yum update** を使用して、CentOS リポジトリからサードパーティのセキュリティ脆弱性修正を直接取得することをお勧めします。これにより、Cisco NX-OS ソフトウェアのアップデートを待つことなく、更新が利用可能になったときに入手できる柔軟性が得られます。

または、**guestshell update** コマンドを使用すると、既存のゲストシェル rootfs が置き換えられます。カスタマイズとソフトウェアパッケージのインストールは、この新しいゲストシェル rootfs のコンテキスト内で再度実行する必要があります。

Guest Shell 1.0 からゲストシェル 2.x へのアップグレード

Guest Shell 2.x は、CentOS 7 ルート ファイル システムに基づいています。コンテンツを Guest Shell 1.0 にプルダウンした .conf ファイルやユーティリティのオフボックス リポジトリがあ

る場合は、Guest Shell 2.x で同じ展開手順を繰り返す必要があります。CentOS 7 の違いを考慮して、展開スクリプトを調整する必要がある場合があります。

Guest Shell 2.x

Cisco NX-OS は、デフォルトで自動的に Guest Shell のインストールおよび有効化を行います。ただし、Guest Shell をサポートしない Cisco NX-OS イメージでデバイスがリロードされる場合、既存の Guest Shell が自動的に削除され、%VMAN-2-INVALID_PACKAGE が発行されます。



- (注) 4GB の RAM を搭載したシステムでは、デフォルトでは Guest Shell が有効になりません。**guestshell enable** コマンドを使用して、Guest Shell をインストールして有効にします。

install all コマンドは、現在の Cisco NX-OS イメージとターゲットの Cisco NX-OS イメージとの互換性を検証します。

互換性のないイメージをインストールした場合の出力例を次に示します。

```
switch#
Installer will perform compatibility check first. Please wait.
uri is: /
2014 Aug 29 20:08:51 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE:
Successfully activated virtual service 'guestshell+'
Verifying image bootflash:/n9kpregs.bin for boot variable "nxos".
##### 100% -- SUCCESS
Verifying image type.
##### 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
##### 100% -- SUCCESS
Preparing "bios" version info using image bootflash:/.
##### 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
##### 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
##### 100% -- SUCCESS
Preparing "nxos" version info using image bootflash:/.
##### 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
##### 100% -- SUCCESS
Preparing "" version info using image bootflash:/.
##### 100% -- SUCCESS
"Running-config contains configuration that is incompatible with the new image (strict
incompatibility).
Please run 'show incompatibility-all nxos <image>' command to find out which feature
needs to be disabled."
Performing module support checks.
##### 100% -- SUCCESS
Notifying services about system upgrade.
[# ] 0% -- FAIL.
Return code 0x42DD0006 ((null)).
"Running-config contains configuration that is incompatible with the new image (strict
incompatibility).
Please run 'show incompatibility-all nxos <image>' command to find out
which feature needs to be disabled."
Service "vman" in vdc 1: Guest shell not supported, do 'guestshell destroy' to remove
it and then retry ISSU
Pre-upgrade check failed. Return code 0x42DD0006 ((null)).
switch#
```



- (注) ベストプラクティスとして、Guest Shell をサポートしていない古い Cisco Nexus イメージをリロードする前に、**guestshell destroy** コマンドを使用して Guest Shell を削除します。

事前設定された SSHD サービス

Guest Shell は、起動時に OpenSSH サーバーを開始します。サーバーは、localhost IP アドレス インターフェイス 127.0.0.1 でランダムに生成されたポートでのみリスンします。これにより、**guestshell** キーワードが入力されたときに、NX-OS **vegas-shell** から Guest Shell へのパスワードなしの接続が提供されます。このサーバーが強制終了されるか、その構成 (/etc/ssh/sshd_config-cisco にある) が変更された場合、NX-OS CLI からの Guest Shell へのアクセスが機能しない可能性があります。

次の手順では、Guest Shell 内で **root** として OpenSSH サーバーをインスタンス化します。

1. SSH 接続を確立するネットワーク名前空間または VRF を決定します。
2. OpenSSH がリスンするポートを決定します。すでに使用されているポートを表示するには、NX-OS コマンドの **show socket connection** を使用します。



- (注) パスワードなしのアクセス用の Guest Shell sshd サービスは、17680 から 49150 までのランダム化されたポートを使用します。ポートの競合を避けるには、この範囲外のポートを選択してください。

次の手順では、OpenSSH サーバーを起動します。例では、IP アドレス 10.122.84.34:2222 で管理 **netns** の OpenSSH サーバーを起動します。

1. 次のファイルを作成します: /usr/lib/systemd/system/sshd-mgmt.service および /etc/ssh/sshd-mgmt_config。ファイルには次の構成が必要です。

```
-rw-r--r-- 1 root root 394 Apr 7 14:21 /usr/lib/systemd/system/sshd-mgmt.service
-rw----- 1 root root 4478 Apr 7 14:22 /etc/ssh/sshd-mgmt_config
```

2. Unit と Service の内容を /usr/lib/systemd/system/ssh.service ファイルから sshd-mgmt.service にコピーします。

3. sshd-mgmt.service ファイルを次のように編集します。

```
[Unit]
Description=OpenSSH server daemon
After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]
EnvironmentFile=/etc/sysconfig/ssh
ExecStartPre=/usr/sbin/sshd-keygen
ExecStart=/sbin/ip netns exec management /usr/sbin/sshd -f /etc/ssh/sshd-mgmt_config
-D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
```

```
RestartSec=42s
[Install]
WantedBy=multi-user.target
```

4. /etc/ssh/sshd-config の内容を /etc/ssh/sshd-mgmt_config にコピーします。必要に応じて、ListenAddress IP とポートを変更します。

```
Port 2222
ListenAddress 10.122.84.34
```

5. 次のコマンドを使用して、systemctl デーモンを開始します。

```
sudo systemctl daemon-reload
sudo systemctl start sshd-mgmt.service
sudo systemctl status sshd-mgmt.service -l
```

6. (オプション) 構成を確認します。

```
ss -tnldp | grep 2222
```

7. ゲストシェルへの SSH :

```
ssh -p 2222 guestshell@10.122.84.34
```

8. 複数の Guest Shell またはスイッチの再起動にまたがって構成を保存します。

```
sudo systemctl enable sshd-mgmt.service
```

9. パスワードなしの SSH/SCP およびリモート実行の場合、ssh-keygen -t dsa コマンドを使用して、SSH/SCP に使用するユーザー ID の公開鍵と秘密鍵を生成します。

その後、キーは /.ssh ディレクトリの id_rsa および id_rsa.pub ファイルに保存されます。

```
[root@node01 ~]# cd ~/.ssh
[root@node02 .ssh]# ls -l
total 8
-rw-----. 1 root root 1675 May 5 15:01 id_rsa
-rw-r--r--. 1 root root 406 May 5 15:01 id_rsa.pub
```

10. 公開キーを SSH で接続するマシンにコピーし、アクセス許可を修正します。

```
cat id_rsa.pub >> /root/.ssh/authorized_keys
chmod 700 /root/.ssh
chmod 600 /root/.ssh/*
```

11. パスワードなしでリモートスイッチに SSH または SCP :

```
ssh -p <port#> userid@hostname [<remote command>]
scp -P <port#> userid@hostname/filepath /destination
```

localtime

Guest Shell は、ホストシステムと /etc/localtime を共有します。



(注) ホストと同じ localtime を共有したくない場合は、このシンボリックリンクを切断して、ゲストシェル固有の /etc/localtime を作成できます。

```
switch(config)# clock timezone PDT -7 0
switch(config)# clock set 10:00:00 27 Jan 2017
Fri Jan 27 10:00:00 PDT 2017
switch(config)# show clock
10:00:07.554 PDT Fri Jan 27 2017
switch(config)# run guestshell
guestshell:~$ date
Fri Jan 27 10:00:12 PDT 2017
```

Guest Shell へのアクセス

Cisco NX-OS のデフォルトでは、network-admin ユーザーのみが Guest Shell にアクセスできます。これはシステムで自動的に有効になっており、**run guestshell** コマンドを使用してアクセスできます。**run bash** コマンドと一致して、これらのコマンドは、NX-OS CLI コマンドの **run guestshell** コマンド形式を使用して Guest Shell 内で発行できます。



(注) Guest Shell は、4 GB を超える RAM を搭載したシステムで自動的に有効になります。

```
switch# run guestshell ls -al /bootflash/*.ova
-rw-rw-rw- 1 2002 503 83814400 Aug 21 18:04 /bootflash/pup.ova
-rw-rw-rw- 1 2002 503 40724480 Apr 15 2012 /bootflash/red.ova
```



(注) 2.2(0.2) 以降の Guest Shell は、スイッチにログインしているユーザーと同じユーザー アカウントを動的に作成します。ただし、他のすべての情報は、スイッチと Guest Shell のユーザー アカウント間で共有されません。

さらに、Guest Shell アカウントは自動的に削除されないため、不要になったときにネットワーク管理者が削除する必要があります。

ゲスト シェルに使用されるリソース

デフォルトでは、ゲストシェルのリソースは、通常のスイッチ操作に使用できるリソースに小さな影響を与えます。ネットワーク管理者がゲスト シェルに追加のリソースを必要とする場合、**guestshell resize {cpu | memory | rootfs}** コマンドは、これらの制限を変更します

リソース	デフォルト	最小/最大
CPU	1 %	1/20%
メモリ	400 MB	256/3840 MB
ストレージ	200 MB	200/2000 MB

CPU 制限は、システム内の他のコンピューティング負荷との競合がある場合に、ゲストシェル内で実行されているタスクに与えられるシステム コンピューティング キャパシティのパーセンテージです。CPU リソースの競合がない場合、ゲストシェル内のタスクは制限されません。



(注) リソース割り当てを変更した後は、ゲストシェルの再起動が必要です。そのために、**guestshell reboot** コマンドを使用できます。

ゲストシェルの機能

Guestshell には、デフォルトで利用可能な多くのユーティリティと機能があります。

ゲストシェルは CentOS 7 Linux 環境であり、この流通向けにビルドされたソフトウェアパッケージを、dnf インストールすることができます。Guestshell には、**net-tools**、**iproute**、**tcpdump** と **OpenSSH** などのネットワークング デバイスで自然に期待される多くの一般的なツールが事前に入力されています。Guestshell 2.x の場合、追加の **python** パッケージをインストールするための PIP と同様に、**python 2.7.5** がデフォルトで含まれています。Guestshell 2.11 では、デフォルトで **python 3.6** も含まれています。

デフォルトでは、ゲストシェルは 64 ビットの実行スペースです。32 ビットのサポートが必要な場合は、**glibc.i686** パッケージを dnf でインストールできます。

Guestshell は、スイッチの管理ポートとデータポートを表すために使用される Linux ネットワーク インターフェイスにアクセスできます。**ifconfig** と **ethtool** などの典型的な Linux のメソッドとユーティリティは、カウンターの収集に使用できます。インターフェイスが NX-OS CLI で VRF に配置されると、Linux ネットワーク インターフェイスはその VRF のネットワーク名前空間に配置されます。名前空間は `/var/run/netns` で見ることができ、**ip netns** ユーティリティを使用してさまざまな名前空間のコンテキストで実行できます。いくつかのユーティリティ、**chvrf** と **vrfinfo** は、別の名前空間で実行し、プロセスが実行されている名前空間 `/vrf` に関する情報を取得するために提供されています。

systemd は、ゲストシェルを含む CentOS 8 環境でサービスを管理するために使用されます。

Guest Shell の NX-OS CLI

ゲストシェルは、ユーザーがゲストシェル環境からホスト ネットワーク要素に NX-OS コマンドを発行できるようにするアプリケーションを提供します。**dohost** アプリケーションは、有効な NX-OS 構成または **exec** コマンドを受け入れ、それらをホスト ネットワーク要素に発行します。

dohost コマンドを呼び出すときは、各 NX-OS コマンドを一重引用符または二重引用符で囲むことができます：

```
dohost "<NXOS CLI>"
```


NX-OS CLI は連鎖させることができます :

```
[guestshell@guestshell ~]$ dohost "sh lldp time | in Hold" "show cdp global"
Holdtime in seconds: 120
Global CDP information:
CDP enabled globally
Refresh time is 21 seconds
Hold time is 180 seconds
CDPv2 advertisements is enabled
DeviceID TLV in System-Name(Default) Format
[guestshell@guestshell ~]$
```

NX-OS CLI は、各コマンドの間にセミコロンを追加することにより、NX-OS スタイルのコマンドチェーン技術を使用して一緒にチェーンすることもできます。(セミコロンの両側にスペースが必要です。) :

```
[guestshell@guestshell ~]$ dohost "conf t ; cdp timer 13 ; show run | inc cdp"
Enter configuration commands, one per line. End with CNTL/Z.
cdp timer 13
[guestshell@guestshell ~]$
```



(注) を使用するリリース 7.0(3)I5(2) の場合、**dohost** コマンドを介してホストで発行されたコマンドは、ゲストシェルユーザの有効なロールに基づく特権で実行されます。

以前のバージョンのゲストシェルは、ネットワーク管理者レベルの権限でコマンドを実行します。

NX-API への UDS 接続の数が最大許容数に達すると、**dohost** コマンドは機能不全になります。

Guest Shell でのネットワーク アクセス

NX-OS スイッチポートは、Guest Shell では Linux ネットワーク インターフェイスとして表されます。ifconfig または ethtool を使用して、/proc/net/dev の表示統計などの一般的な Linux メソッドはすべてサポートされています。

Guest Shell には、多くの一般的なネットワークユーティリティがデフォルトで含まれており、**chvrf vrf command** コマンドを使用してさまざまな VRF で使用できます。

```
[guestshell@guestshell bootflash]$ ifconfig Eth1-47
Eth1-47: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 13.0.0.47 netmask 255.255.255.0 broadcast 13.0.0.255
ether 54:7f:ee:8e:27:bc txqueuelen 100 (Ethernet)
RX packets 311442 bytes 21703008 (20.6 MiB)
RX errors 0 dropped 185 overruns 0 frame 0
TX packets 12967 bytes 3023575 (2.8 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Guest Shell 内では、ネットワーク状態をモニタリングできますが、変更することはできません。ネットワーク状態を変更するには、ホストの bash シェルで NX-OS CLI または適切な Linux ユーティリティを使用します。

この **tcpdump** コマンドはGuest Shell にパッケージ化されており、管理ポートまたはスイッチポートでパントされたトラフィックのパケットトレースを可能にします。

この **sudo ip netns exec management ping** ユーティリティは、指定されたネットワーク名前空間のコンテキストでコマンドを実行するための一般的な方法です。これはGuest Shell 内で実行できます。

```
[guestshell@guestshell bootflash]$ sudo ip netns exec management ping 10.28.38.48
PING 10.28.38.48 (10.28.38.48) 56(84) bytes of data.
64 bytes from 10.28.38.48: icmp_seq=1 ttl=48 time=76.5 ms
```

chvrf ユーティリティは便宜のために提供されています。

```
guestshell@guestshell bootflash]$ chvrf management ping 10.28.38.48
PING 10.28.38.48 (10.28.38.48) 56(84) bytes of data.
64 bytes from 10.28.38.48: icmp_seq=1 ttl=48 time=76.5 ms
```



(注) コマンドなしで実行される **chvrf** コマンドは、現在の VRF / ネットワーク名前空間で実行されます。

たとえば、管理 VRF 経由で IP アドレス 10.0.0.1 を ping するには、コマンドは「**chvrf management ping 10.0.0.1**」です。 **scp** または **ssh** などの他のユーティリティも同様です。

例：

```
switch# guestshell
[guestshell@guestshell ~]$ cd /bootflash
[guestshell@guestshell bootflash]$ chvrf management scp foo@10.28.38.48:/foo/index.html
index.html
foo@10.28.38.48's password:
index.html 100% 1804 1.8KB/s 00:00
[guestshell@guestshell bootflash]$ ls -al index.html
-rw-r--r-- 1 guestshe users 1804 Sep 13 20:28 index.html
[guestshell@guestshell bootflash]$
[guestshell@guestshell bootflash]$ chvrf management curl cisco.com
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://www.cisco.com/">here</a>.</p>
</body></html>
[guestshell@guestshell bootflash]$
```

システム上の VRF のリストを取得するには、NX-OS からネイティブに、**show vrf** または **dohost** コマンドを介してコマンドを使用します。

例：

```
[guestshell@guestshell bootflash]$ dohost 'sh vrf'
VRF-Name  VRF-ID  State  Reason
default   1         Up     --
management 2         Up     --
```

```
red          6          Up          --
```

Guest Shell内では、VRFに関連付けられたネットワーク名前空間が実際に使用されます。どのネットワーク名前空間が存在するかを確認の方が便利な場合があります。

```
[guestshell@guestshell bootflash]$ ls /var/run/netns
default management red
[guestshell@guestshell bootflash]$
```

Guest Shell内からドメイン名を解決するには、リゾルバーを構成する必要があります。Guest Shellで/etc/resolv.confファイルを編集して、ネットワークに適したDNSネームサーバとドメインを含めます。

例：

```
nameserver 10.1.1.1
domain cisco.com
```

ネームサーバとドメインの情報は、NX-OS構成で構成されたものと一致する必要があります。

例：

```
switch(config)# ip domain-name cisco.com
switch(config)# ip name-server 10.1.1.1
switch(config)# vrf context management
switch(config-vrf)# ip domain-name cisco.com
switch(config-vrf)# ip name-server 10.1.1.1
```

スイッチがHTTPプロキシサーバーを使用するネットワーク内にある場合、**http_proxy** および **https_proxy** 環境変数も Guest Shell内で設定する必要があります。

例：

```
export http_proxy=http://proxy.esl.cisco.com:8080
export https_proxy=http://proxy.esl.cisco.com:8080
```

これらの環境変数は、.bashrcファイルまたは適切なスクリプトで設定して、永続的であることを確認する必要があります。

ゲストシェルでのブートフラッシュへのアクセス

ネットワーク管理者は、NX-OS CLI コマンドの使用に加えて、Linux コマンドとユーティリティを使用してファイルを管理できます。ゲストシェル環境の/bootflashにシステムブートフラッシュをマウントすることにより、network-adminはLinuxコマンドを使用してこれらのファイルを操作できます。

例：

```
find . -name "foo.txt"
```

```
rm "/bootflash/junk/foo.txt"
```

Guest Shell の Python

Python はインタラクティブに使用できますが、python スクリプトをゲストシェルで実行することもできます。

例：

```
guestshell:~$ python
Python 2.7.5 (default, Jun 24 2015, 00:41:19)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
guestshell:~$
```

ネットワーク管理者が新しい Python パッケージをインストールできるように、ゲストシェルには pip python パッケージマネージャが含まれています。

例：

```
[guestshell@guestshell ~]$ sudo su
[root@guestshell guestshell]# pip install Markdown
Collecting Markdown
Downloading Markdown-2.6.2-py2.py3-none-any.whl (157kB)
100% |#####| 159kB 1.8MB/s
Installing collected packages: Markdown
Successfully installed Markdown-2.6.2
[root@guestshell guestshell]# pip list | grep Markdown
Markdown (2.6.2)
[root@guestshell guestshell]#
```



(注) **pip install** コマンドを入力する前に、**sudo su** コマンドを入力する必要があります。

Guest Shell バージョン 2.10 までの Python 3 (CentOS 7)

ゲストシェル 2.X は、デフォルトで Python 3 がインストールされていない CentOS 7.1 環境を提供します。CentOS 7.1 に Python 3 をインストールするには、サードパーティのリポジトリを使用する、送信元からビルドするなど、複数の方法があります。別のオプションは、同じシステム内に複数のバージョンの Python のインストールをサポートする Red Hat Software Collections を使用することです。

Red Hat Software Collections (SCL) ツールをインストールするには:

1. scl-utils パッケージをインストールします。
2. CentOS SCL リポジトリを有効にして、提供されている Python 3 RPM のいずれかをインストールします。

```
[admin@guestshell ~]$ sudo su
[root@guestshell admin]# yum install -y scl-utils | tail
Running transaction test
Transaction test succeeded
Running transaction
  Installing : scl-utils-20130529-19.el7.x86_64                1/1
  Verifying  : scl-utils-20130529-19.el7.x86_64                1/1

Installed:
  scl-utils.x86_64 0:20130529-19.el7

Complete!

[root@guestshell admin]# yum install -y centos-release-scl | tail
Verifying : centos-release-scl-2-3.el7.centos.noarch          1/2
Verifying : centos-release-scl-rh-2-3.el7.centos.noarch        2/2

Installed:
  centos-release-scl.noarch 0:2-3.el7.centos

Dependency Installed:
  centos-release-scl-rh.noarch 0:2-3.el7.centos

Complete!

[root@guestshell admin]# yum install -y rh-python36 | tail
warning: /var/cache/yum/x86_64/7/centos-scl-rh/packages/rh-python36-2.0-1.el7.x86_64.rpm:
Header V4 RSA/SHA1 Signature, key ID f2ee9d55: NOKEY
http://centos.sonn.com/7.7.1908/os/x86_64/Packages/groff-base-1.22.2-8.el7.x86_64.rpm:
[Errno 12] Timeout on
http://centos.sonn.com/7.7.1908/os/x86_64/Packages/groff-base-1.22.2-8.el7.x86_64.rpm:
(28, 'Operation too slow. Less than 1000 bytes/sec transferred the last 30 seconds')
Trying other mirror.
Importing GPG key 0xF2EE9D55:
  Userid      : "CentOS SoftwareCollections SIG
(https://wiki.centos.org/SpecialInterestGroup/SCLO) <security@centos.org>"
  Fingerprint: c4db d535 b1fb ba14 f8ba 64a8 4eb8 4e71 f2ee 9d55
  Package     : centos-release-scl-rh-2-3.el7.centos.noarch (@extras)
  From        : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-SIG-SCLO
  rh-python36-python-libs.x86_64 0:3.6.9-2.el7
  rh-python36-python-pip.noarch 0:9.0.1-2.el7
  rh-python36-python-setuptools.noarch 0:36.5.0-1.el7
  rh-python36-python-virtualenv.noarch 0:15.1.0-2.el7
  rh-python36-runtime.x86_64 0:2.0-1.el7
  scl-utils-build.x86_64 0:20130529-19.el7
  xml-common.noarch 0:0.6.3-39.el7
  zip.x86_64 0:3.0-11.el7
```

Complete!

SCL を使用すると、Python 3 の環境変数を自動的に設定して、インタラクティブな bash セッションを作成できます。



(注) SCL Python インストールを使用するためにルート ユーザーは必要ありません。

```
[admin@guestshell ~]$ scl enable rh-python36 bash
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python SCL のインストールでは、pip ユーティリティも提供されます。

```
[admin@guestshell ~]$ pip3 install requests --user
Collecting requests
  Downloading
https://files.pythonhosted.org/packages/51/7d/23c926cc341ee6b7c102a00ba99a0f828e89c72b2190e27c11d4b7fb/requests-2.22.0-py2.py3-none-any.whl
(57kB)
  100% |#####| 61kB 211kB/s
Collecting idna<2.9,>=2.5 (from requests)
  Downloading
https://files.pythonhosted.org/packages/14/2c/c551d81dce15200belcf41cd03869a46fe7226e7450af7a6543fc474c9/idna-2.8-py2.py3-none-any.whl
(58kB)
  100% |#####| 61kB 279kB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Downloading
https://files.pythonhosted.org/packages/bc/a9/01ffefbf562e4274c6487b431ddc7ca55c7510b22e451f1409844368/chardet-3.0.4-py2.py3-none-any.whl
(133kB)
  100% |#####| 143kB 441kB/s
Collecting certifi>=2017.4.17 (from requests)
  Downloading
https://files.pythonhosted.org/packages/b9/63/d50cc99a0fb00655a3993f1db8a75e24b7890c9cfd8e99/certifi-2019.11.28-py2.py3-none-any.whl
(156kB)
  100% |#####| 163kB 447kB/s
Collecting urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 (from requests)
  Downloading
https://files.pythonhosted.org/packages/e8/74/6e4f91745020f967cd9332b2889b10090957334692d68e44afe91b77e/urllib3-1.25.8-py2.py3-none-any.whl
(125kB)
  100% |#####| 133kB 656kB/s
Installing collected packages: idna, chardet, certifi, urllib3, requests
Successfully installed certifi-2019.11.28 chardet-3.0.4 idna-2.8 requests-2.22.0
urllib3-1.25.8
You are using pip version 9.0.1, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
[admin@guestshell ~]$ python3
Python 3.6.9 (default, Nov 11 2019, 11:24:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> requests.get("https://cisco.com")
<Response [200]>
```

デフォルトの Python 2 インストールは、SCL Python インストールと一緒に使用できます。

```
[admin@guestshell ~]$ which python3
/opt/rh/rh-python36/root/usr/bin/python3
[admin@guestshell ~]$ which python2
/bin/python2
[admin@guestshell ~]$ python2
Python 2.7.5 (default, Aug 7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Hello world!'
Hello world!
```

Software Collections を使用すると、同じ RPM の複数のバージョンをシステムにインストールできます。この場合、Python 3.6 に加えて Python 3.5 をインストールすることが可能です。

```
[admin@guestshell ~]$ sudo yum install -y rh-python35 | tail
Dependency Installed:
rh-python35-python.x86_64 0:3.5.1-13.e17
rh-python35-python-devel.x86_64 0:3.5.1-13.e17
rh-python35-python-libs.x86_64 0:3.5.1-13.e17
rh-python35-python-pip.noarch 0:7.1.0-2.e17
rh-python35-python-setuptools.noarch 0:18.0.1-2.e17
```

```

rh-python35-python-virtualenv.noarch 0:13.1.2-2.el7
rh-python35-runtime.x86_64 0:2.0-2.el7

Complete!

[admin@guestshell ~]$ scl enable rh-python35 python3
Python 3.5.1 (default, May 29 2019, 15:41:33)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>

```



(注) 複数の Python バージョンが SCL にインストールされているときに新しいインタラクティブ bash セッションを作成すると、libpython 共有オブジェクトファイルをロードできないという問題が発生する可能性があります。 **source scl_source enable python-installation** コマンドを使用して、現在の bash セッションで環境を適切にセットアップできる回避策があります。

デフォルトの Guest Shell ストレージのキャパシティが、Python 3 をインストールするのに十分ではありません。 **guestshell resize rootfs size-in-MB** コマンドを使用して、ファイルシステムのサイズを増やします。通常、rootfs のサイズを 550 MB に設定すれば十分です。

Installing RPMs in the Guest Shell

The `/etc/yum.repos.d/CentOS-Base.repo` file is set up to use the CentOS mirror list by default. Follow instructions in that file if changes are needed.

Yum can be pointed to one or more repositories at any time by modifying the `yumrepo_x86_64.repo` file or by adding a new `.repo` file in the `repos.d` directory.

For applications to be installed inside Guest Shell 2.x, go to the CentOS 7 repo at http://mirror.centos.org/centos/7/os/x86_64/Packages/.

Yum resolves the dependencies and installs all the required packages.

```

[guestshell@guestshell ~]$ sudo chvrf management yum -y install glibc.i686
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
* base: bay.uchicago.edu
* extras: pubmirrors.dal.coreospace.com
* updates: mirrors.cmich.edu
Resolving Dependencies
"-->" Running transaction check
"--->" Package glibc.i686 0:2.17-78.el7 will be installed
"-->" Processing Dependency: libfreebl3.so(NSSRAWHASH_3.12.3) for package:
glibc-2.17-78.el7.i686
"-->" Processing Dependency: libfreebl3.so for package: glibc-2.17-78.el7.i686
"-->" Running transaction check
"--->" Package nss-softokn-freebl.i686 0:3.16.2.3-9.el7 will be installed
"-->" Finished Dependency Resolution

```

Dependencies Resolved

Package	Arch	Version	Repository	Size
---------	------	---------	------------	------

```

Installing:
glibc i686 2.17-78.el7 base 4.2 M
Installing for dependencies:

```

```

nss-softokn-freebl i686 3.16.2.3-9.el7 base 187 k

Transaction Summary
-----
Install 1 Package (+1 Dependent package)

Total download size: 4.4 M
Installed size: 15 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
(1/2): nss-softokn-freebl-3.16.2.3-9.el7.i686.rpm | 187 kB 00:00:25
(2/2): glibc-2.17-78.el7.i686.rpm | 4.2 MB 00:00:30

-----
Total 145 kB/s | 4.4 MB 00:00:30
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : nss-softokn-freebl-3.16.2.3-9.el7.i686 1/2
Installing : glibc-2.17-78.el7.i686 2/2
error: lua script failed: [string "%triggerin(glibc-common-2.17-78.el7.x86_64)":1:
attempt to compare number with nil
Non-fatal "<"unknown">" scriptlet failure in rpm package glibc-2.17-78.el7.i686
Verifying : glibc-2.17-78.el7.i686 1/2
Verifying : nss-softokn-freebl-3.16.2.3-9.el7.i686 2/2

Installed:
glibc.i686 0:2.17-78.el7

Dependency Installed:
nss-softokn-freebl.i686 0:3.16.2.3-9.el7

Complete!

```



Note When more space is needed in the Guest Shell root file system for installing or running packages, the **guestshell resize roots *size-in-MB*** command is used to increase the size of the file system.



Note Some open source software packages from the repository might not install or run as expected in the Guest Shell as a result of restrictions that have been put into place to protect the integrity of the host system.

仮想サービスのセキュリティ ポスチャ

スイッチでのゲストシェルと仮想サービスの使用は、ネットワーク管理者がシステムの機能を管理または拡張できる多くの方法のうち2つにすぎません。これらのオプションは、ネイティブホストコンテキストから切り離された実行環境を提供することを目的としています。この分離により、ネイティブの実行環境と互換性がない可能性のあるソフトウェアをシステムに導入できます。また、システムの動作、パフォーマンス、またはスケールに影響を与えない環境でソフトウェアを実行することもできます。

[デジタル署名されたアプリケーションパッケージ (Digitally Signed Application Packages)]

デフォルトでは、Cisco ネットワーク要素は、アプリケーションのランタイムに有効な Cisco デジタル署名を提供することを要求します。Cisco のデジタル署名により、Cisco が開発したパッケージとアプリケーションの完全性が保証されます。

Cisco Nexus 9000 シリーズ スイッチは、署名されていない OVA ソフトウェア パッケージを許可する署名レベルポリシーの構成をサポートしています。署名されていないパッケージと Cisco 署名されたパッケージで仮想サービスを作成できるようにするために、ネットワーク管理者は次を構成できます。

```
virtual-service
  signing level unsigned
```



(注) ゲストシェルソフトウェアパッケージには Cisco の署名があり、この構成は必要ありません。

[カーネル脆弱性パッチ (Kernel Vulnerability Patches)]

シスコは、既知の脆弱性に対処するプラットフォーム アップデートで、関連する Common Vulnerabilities and Exposures (CVE) に対応します。

[ASLR および X-Space のサポート (ASLR and X-Space Support)]

Cisco 9000 NX-OS は、ランタイム ディフェンスのためのアドレス空間 Layout Randomization (ASLR) と Executable Space Protection (X-Space) の使用をサポートしています。Cisco が署名したパッケージのソフトウェアは、この機能を利用します。システムに他のソフトウェアがインストールされている場合は、これらのテクノロジーをサポートするホスト OS と開発ツールチェーンを使用して構築することをお勧めします。これにより、ソフトウェアが潜在的な侵入者に提示する潜在的な攻撃対象領域が減少します。

名前空間の分離

ホストと仮想サービスは、別々の名前空間に分けられます。これは、仮想サービスの実行スペースをホストから分離する基礎を提供します。名前空間の分離は、信頼境界間の偶発的または意図的なデータの上書きによるデータの損失やデータの破損から保護するのに役立ちます。また、機密データ漏洩を防止することにより、機密データの完全性を確保するのに役立ちます。ある仮想サービスのアプリケーションは、別の仮想サービスのデータにアクセスできません。

ルータユーザーの制限

安全なコードを開発するためのベストプラクティスとして、割り当てられたタスクを実行するために必要な最小限の特権でアプリケーションを実行することを推奨します。意図しないアクセスを防ぐために、Guest Shellに追加されたソフトウェアは、このベストプラクティスに従う必要があります。

仮想サービスはLinuxの機能が低下したことによる制限の対象となります。アプリケーションでroot権限を必要とする操作を実行する必要がある場合は、rootアカウントの使用を、rootアクセスが絶対に必要な最小限の操作セットに制限し、そのモードでアプリケーションを実行できる時間のハード制限などの他の制御を課します。

内のルートに対してドロップされる一連のLinux機能は次のとおりです。

CAP_SYS_BOOT	CAP_MKNOD	CAP_SYS_PACCT
CAP_SYS_MODULE	CAP_MAC_OVERRIDE	CAP_SYS_RESOURCE
CAP_SYS_TIME	CAP_SYS_RAWIO	CAP_AUDIT_WRITE
CAP_AUDIT_CONTROL	CAP_SYS_NICE	CAP_NET_ADMIN
CAP_MAC_ADMIN	CAP_SYS_PTRACE	

仮想サービス内のルートとして、tmpfsとramfsマウントだけでなくバインドマウントも使用できます。他のマウントは防止されます。

リソース管理

DDoS攻撃は、攻撃対象のユーザがマシンやネットワーク技術情報を使用できないようにする試みます。不適切な動作または悪意のあるアプリケーションコードは、接続帯域幅、ディスク容量、メモリ、およびその他のリソースの過剰消費の結果としてDoSを引き起こす可能性があります。ホストは、で技術情報を公平に割り当てる技術情報管理機能を提供します。

ゲストファイルシステムのアクセス制限

仮想サービス内のファイルの完全性を維持するために、仮想サービスのファイルシステムにはNX-OS CLIからアクセスできません。特定の仮想サービスがファイルの変更を許可している場合、これを実行できる代替手段(つまりyum install、scp、ftpなど)を提供する必要があります。

ホストの[ブートフラッシュ:(bootflash:)]と[揮発性:(volatile:)]は、ゲストシェル内で[/ブートフラッシュ(/bootflash)]および[/揮発性(/volatile)]としてマウントされます。ネットワーク管理者は、ホストからNX-OS execコマンドを使用するか、ゲストシェル内からLinuxコマンドを使用して、このメディア上のファイルにアクセスできます。

ゲストシェルの管理

以下は、ゲストシェルを管理するためのコマンドです。

表 1: ゲストシェル CLI コマンド

コマンド	説明
<p>guestshell enable {package [<i>guest shell OVA file</i> <i>rootfs-file-URI</i>]}</p>	<ul style="list-style-type: none"> • [ゲストシェル OVA ファイル (<i>guest shell OVA file</i>)] 指定時 : <p>システムイメージに組み込まれている OVA を使用して、ゲストシェルをインストールしてアクティブ化します。</p> <p>指定されたソフトウェア パッケージ (OVA ファイル) またはシステムイメージからの組み込みパッケージ (パッケージが指定されていない場合) を使用して、ゲストシェルをインストールしてアクティブ化します。当初、ゲスト シェル パッケージは、システムイメージに埋め込むことによるのみ利用できます。</p> <p>ゲストシェルがすでにインストールされている場合、このコマンドはインストールされているゲストシェルを有効にします。通常、これは guestshell disable コマンドの後に使用されます。</p> • <i>rootfs-file-URI</i> が指定されている場合 : <p>ゲストシェルが破棄された状態のときに、ゲスト シェル rootfs をインポートします。このコマンドは、指定されたパッケージでゲスト シェルを起動します。</p>
<p>guestshell export rootfs package <i>destination-file-URI</i></p>	<p>ゲストシェルの rootfs ファイルをローカル URI (ブートフラッシュ、USB1 など) にエクスポートします。 [(7.0 (3) I7 (1) 以降のリリース)]</p>
<p>guestshell disable</p>	<p>シャットダウンとゲストシェルの無効化</p>

コマンド	説明
<p>guestshell upgrade {package [<i>guest shell OVA file</i> <i>rootfs-file-URI</i>]}</p>	<ul style="list-style-type: none"> • [<i>ゲストシェルOVAファイル (guest shell OVA file)</i>] 指定時 : <p>指定されたソフトウェア パッケージ (OVA ファイル) またはシステムイメージからの組み込みパッケージ (パッケージが指定されていない場合) を使用して、ゲスト シェルを非アクティブ化してアップグレードします。当初、ゲスト シェル パッケージは、システム イメージに埋め込むことによつてのみ利用できます。</p> <p>ゲスト シェルの現在の rootfs は、ソフトウェア パッケージの rootfs に置き換えられます。ゲスト シェルは、アップグレード後も持続するセカンダリ ファイル システムを利用しません。永続的なセカンダリ ファイル システムがない場合、guestshell destroy コマンドに続けて guestshell enable コマンドを使用して rootfs を置き換えることもできます。アップグレードが成功すると、ゲスト シェルがアクティブ化されます。</p> <p>アップグレード コマンドを実行する前に、確認を求めるプロンプトが表示されます。</p> • <i>rootfs-file-URI</i> が指定されている場合 : <p>ゲスト シェルがすでにインストールされている場合、ゲスト シェルの rootfs ファイルをインポートします。このコマンドは、既存のゲスト シェルを削除します。</p> <p>指定されたパッケージにインストールします。</p>

コマンド	説明
<p>guestshell reboot</p>	<p>ゲストシェルを非アクティブ化してから、再度アクティブ化します。</p> <p>リブート コマンドを実行する前に、確認を求めるプロンプトが表示されます。</p> <p>(注) これは、exec モードで guestshell disable コマンドの後に guestshell enable コマンドが続くのと同じです。</p> <p>これは、ゲストシェル内のプロセスが停止しており、再起動する必要がある場合に役立ちます。この run guestshell コマンドは、ゲストシェルで実行されている <code>sshd</code> に依存しています。</p> <p>コマンドが機能しない場合は、<code>sshd</code> プロセスが誤って停止した可能性があります。NX-OS CLI からゲストシェルの再起動を実行すると、再起動してコマンドを復元できます。</p>
<p>guestshell destroy</p>	<p>ゲストシェルサービスを非アクティブ化して、アンインストールします。ゲストシェルに関連付けられているすべての技術情報がシステムに返されます。この show virtual-service global コマンドは、これらの技術情報がいつ利用可能になるかを示します。</p> <p>このコマンドを発行すると、<code>destroy</code> コマンドを実行する前に確認を求めるプロンプトが表示されます。</p>
<p>guestshell run guestshell</p>	<p>シェルプロンプトですでに実行されているゲストシェルに接続します。必要なユーザー名/パスワード</p>
<p>guestshell run <i>command</i> run guestshell <i>command</i></p>	<p>ゲストシェル環境のコンテキスト内で Linux / UNIX コマンドを実行します。</p> <p>コマンドの実行後、スイッチプロンプトに戻ります。</p>

コマンド	説明
guestshell resize [cpu memory rootfs]	<p>ゲストシェルに割り当てられた使用可能な技術情報を変更します。変更は、次にゲストシェルが有効化または再起動されたときに有効になります。</p> <p>(注) サイズ変更の値は、guestshell destroy コマンドを使用するとクリアされます。</p>
guestshell sync	<p>アクティブスーパーバイザとスタンバイスーパーバイザがあるシステムでは、このコマンドはゲストシェルの格納ファイルをアクティブスーパーバイザからスタンバイスーパーバイザに同期します。network-admin は、スタンバイスーパーバイザが現用系スーパーバイザになったときに同じ rootfs を使用するようにゲストシェル rootfs が設定されているときに、このコマンドを発行します。このコマンドを使用しない場合、スタンバイスーパーバイザがそのスーパーバイザで利用可能なゲストシェルパッケージを使用してアクティブロールに移行するときに、ゲストシェルが新たにインストールされます。</p>
virtual-service reset force	<p>ゲストシェルまたは仮想サービスを管理できない場合は、システムのリロード後でも、reset コマンドを使用してゲストシェルとすべての仮想サービスを強制的に削除します。クリーンアップを実行するには、システムを再ロードする必要があります。このコマンドを発行した後は、システムがリロードされるまで、ゲストシェルまたは追加の仮想サービスをインストールまたは有効にすることはできません。</p> <p>リセットを開始する前に確認を求められます。</p>



(注) ゲストシェル環境を有効化/無効化し、アクセスするには、管理者権限が必要です。



- (注) ゲストシェルは、ホストシステム上の Linux コンテナ (LXC) として導入されます。NX-OS デバイスでは、LXC は `virtual-service` コマンドでインストールと管理されます。ゲストシェルは、`virtual-service` コマンドに `guestshell+` という名前の仮想サービスとして表示されます。

Guest Shell の無効化

`guestshell disable` コマンドはシャットダウンして、Guest Shell を無効化します。

Guest Shell が無効化された状態でシステムをリロードすると、Guest Shell は無効化されたままになります。

例：

```
switch# show virtual-service list
Virtual Service List:
Name                Status              Package Name
-----
guestshell+         Activated           guestshell.ova
switch# guestshell disable
You will not be able to access your guest shell if it is disabled. Are you sure you want
to disable the guest shell? (y/n) [n] y

2014 Jul 30 19:47:23 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Deactivating virtual
service 'guestshell+'
2014 Jul 30 18:47:29 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully deactivated
virtual service 'guestshell+'
switch# show virtual-service list
Virtual Service List:
Name                Status              Package Name
-----
guestshell+         Deactivated         guestshell.ova
```



- (注) `guestshell enable` コマンドで Guest Shell が再アクティブ化されます。

ゲストシェルの破棄

`guestshell destroy` コマンドは、ゲストシェルとそのアーティファクトをアンインストールします。このコマンドでは、ゲストシェル OVA は削除されません。

ゲストシェルが破棄された状態でシステムをリロードすると、ゲストシェルは破棄されたままになります。

```
switch# show virtual-service list
Virtual Service List:
Name                Status              Package Name
-----
guestshell+         Deactivated         guestshell.ova

switch# guestshell destroy
```

```

You are about to destroy the guest shell and all of its contents. Be sure to save your
work. Are you sure you want to continue? (y/n) [n] y
2014 Jul 30 18:49:10 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Destroying virtual service
'guestshell+'
2014 Jul 30 18:49:10 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Successfully destroyed
virtual service 'guestshell +'

switch# show virtual-service list
Virtual Service List:

```



(注) **guestshell enable** コマンドを使用して、ゲスト シェルを再度有効にすることができます。



(注) ゲスト シェルを使用しない場合は、**guestshell destroy** コマンドで削除できます。ゲスト シェルが削除されると、その後のリロードのために削除されたままになります。つまり、ゲスト シェル コンテナが削除され、スイッチが再ロードされても、ゲスト シェル コンテナは自動的に開始されません。

Guest Shell の有効化

この **guestshell enable** コマンドは、Guest Shell ソフトウェア パッケージから Guest Shell をインストールします。デフォルトでは、システムイメージに埋め込まれたパッケージがインストールに使用されます。Guest Shell が無効化されている場合は、このコマンドを使用して、Guest Shell を再アクティブ化することもできます。

Guest Shell が有効化された状態でシステムをリロードすると、Guest Shell は有効化されたままになります。

例：

```

switch# show virtual-service list
Virtual Service List:
switch# guestshell enable
2014 Jul 30 18:50:27 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual service
'guestshell+'
2014 Jul 30 18:50:42 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating

2014 Jul 30 18:50:42 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual
service 'guestshell+'
2014 Jul 30 18:51:16 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'

switch# show virtual-service list
Virtual Service List:
Name                Status              Package Name
guestshell+         Activated           guestshell.ova

```


ゲストシェルの複製

Cisco NX-OS リリース 7.0 (3) I7 (1) 以降、1つのスイッチでカスタマイズされたゲストシェル **rootfs** を複数のスイッチに展開できます。

アプローチは、ゲストシェル **rootfs** をカスタマイズしてからエクスポートし、ファイルサーバに保存することです。POAP スクリプトは、ゲストシェル **rootfs** を他のスイッチにダウンロード (インポート) し、特定のゲストシェルを多数のデバイスに同時にインストールできます。

ゲストシェル **rootfs** のエクスポート

ゲストシェル **rootfs** をエクスポートするには、**guestshell export rootfs package destination-file-URI** コマンドを使用します。

destination-file-URI パラメータは、ゲストシェル **rootfs** のコピー先のファイルの名前です。このファイルでは、ローカル URI オプション (ブートフラッシュ、USB1 など) が可能です。

guestshell export rootfs package コマンドでは、次の処理が行われます。

- ゲストシェルを無効にします (すでに有効になっている場合)。
- ゲストシェルインポート YAML ファイルを作成し、**rootfs ext4** ファイルの `/cisco` ディレクトリに挿入します。
- **rootfs ext4** ファイルをターゲット URI の場所にコピーします。
- ゲストシェルが以前に有効になっていた場合は、再度有効にします。

Guest Shell **rootfs** のインポート

Guest Shell **rootfs** をインポートする場合、考慮すべき 2 つの状況があります。

- Guest Shell が破棄された状態の場合は、**guestshell enable package rootfs-file-URI** コマンドを使用して、Guest Shell **rootfs** をインポートします。このコマンドは、指定されたパッケージで Guest Shell を起動します。
- Guest Shell がすでにインストールされている場合は、**guestshell upgrade package rootfs-file-URI** コマンドを使用して、Guest Shell **rootfs** をインポートします。このコマンドは、既存の Guest Shell を削除し、指定されたパッケージをインストールします。

rootfs-file-URI パラメータは、ローカルストレージ (ブートフラッシュ、USB など) に保存されている **rootfs** ファイルです。

ブートフラッシュにあるファイルでこのコマンドを実行すると、ファイルはブートフラッシュのストレージプールに移動されます。

ベストプラクティスとして、**guestshell upgrade package rootfs-file-URI** コマンドを使用する前に、ファイルをブートフラッシュにコピーし、`md5sum` を検証する必要があります。



(注) **guestshell upgrade package rootfs-file-URI** コマンドは、Guest Shell 内から実行



(注) rootfs ファイルはシスコの署名付きパッケージではありません。例に示すように、有効にする前に、署名されていないパッケージを許可するように設定する必要があります。

```
(config-virt-serv-global)# signing level unsigned
Note: Support for unsigned packages has been user-enabled. Unsigned packages are not endorsed by Cisco. User assumes all responsibility.
```



(注) rootfs の組み込みバージョンを復元するには：

- Guest Shell が既にインストールされている場合は、**guestshell upgrade** コマンドを（追加のパラメーターなしで）使用します。
- Guest Shell が破棄されたときに、**guestshell enable** コマンドを（追加パラメータなしで）使用します。



(注) Guest Shell 内から、または NX-API を使用してスイッチの外部からこのコマンドを実行する場合は、プロンプトをスキップするように設定する必要があります。**terminal dont-ask**

guestshell enable package rootfs-file-URI コマンド：

- **rootfs** ファイルの基本的な検証を実行します。
- **rootfs** をストレージプールに移動します。
- **rootfs** をマウントして、/cisco ディレクトリから YAML ファイルを抽出します。
- YAML ファイルを解析して VM 定義（リソース要件を含む）を取得します。
- Guest Shell をアクティブにします。

guestshell enable のワークフローの例：

```
switch# copy scp://user@10.1.1.1/my_storage/gs_rootfs.ext4 bootflash: vrf management
switch# guestshell resize cpu 8
Note: System CPU share will be resized on Guest shell enable
switch# guestshell enable package bootflash:gs_rootfs.ext4
Validating the provided rootfs
switch# 2017 Jul 31 14:58:01 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Installing virtual
service 'guestshell+'
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-INSTALL_STATE: Install success virtual
service 'guestshell+'; Activating
2017 Jul 31 14:58:09 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual
```

```
service 'guestshell+'
2017 Jul 31 14:58:33 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully activated
virtual service 'guestshell+'
```



(注) **guestshell upgrade** のワークフローの前に、既存の Guest Shell が破棄されます。



(注) サイズ変更の値は、**guestshell upgrade** コマンドを使用するとクリアされます。

YAML ファイルのインポート

Guest Shell のユーザーが変更可能ないくつかの特性を定義する YAML ファイルは、エクスポート操作の一部として自動的に作成されます。これは、/cisco ディレクトリの **Guest Shell rootfs** に組み込まれています。これは、Guest Shell コンテナの完全な記述子ではありません。ユーザーが変更できるパラメータの一部のみが含まれています。

Guest Shell インポート YAML ファイルの例：

```
---
import-schema-version: "1.0"
info:
  name: "GuestShell"
  version: "2.2(0.3)"
  description: "Exported GuestShell: 20170216T175137Z"
app:
  apptype: "lxc"
  cpuarch: "x86_64"
  resources:
    cpu: 3
    memory: 307200
    disk:
      - target-dir: "/"
        capacity: 250
...
```

guestshell export rootfs package コマンドを実行すると、YAML ファイルが生成されます。このファイルは、現在実行中の Guest Shell の値をキャプチャします。

情報セクションには、Guest Shell の識別に役立つ非運用データが含まれています。**show guestshell detail** コマンドの出力に一部の情報が表示されます。

説明の値は、YAML ファイルが作成されたときの UTC 時間のエンコーディングです。時刻文字列のフォーマットは、RFC5545 (iCal) の DTSTAMP と同じです。

リソースセクションでは、Guest Shell をホストするために必要な情報技術について説明します。この例の **target-dir** の値「/」は、ディスクを **rootfs** として識別します。



(注) Guest Shell が破棄されたときにサイズ変更された値が指定された場合、**guestshell enable package** コマンドの使用時にそれらの値がインポート YAML ファイルの値よりも優先されます。

cpuarch 値は、コンテナの実行が予想される CPU アーキテクチャを示します。

エクスポート操作が完了した後、YAML ファイルを変更できます（説明などを変更したり、必要に応じて技術情報パラメータを増やしたりできます）。

Cisco は、JSON スキーマを使用して変更された YAML ファイルを検証するために実行できる Python スクリプトを提供しています。完全なテストではありませんが（たとえば、デバイス固有のリソース制限はチェックされません）、一般的なエラーにフラグを付けることができます。例を含む Python スクリプトは、[Guest Shell インポートエクスポート (Guest Shell Import Export)] https://github.com/datacenter/opennxd/tree/master/guestshell_import_export にあります。次の JSON ファイルは、Guest Shell インポート YAML のバージョン 1.0 のスキーマを記述しています。

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Guest Shell import schema",
  "description": "Schema for Guest Shell import descriptor file - ver 1.0",
  "copyright": "2017 by Cisco systems, Inc. All rights reserved.",
  "id": "",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "import-schema-version": {
      "id": "/import-schema-version",
      "type": "string",
      "minLength": 1,
      "maxLength": 20,
      "enum": [
        "1.0"
      ]
    },
    "info": {
      "id": "/info",
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "name": {
          "id": "/info/name",
          "type": "string",
          "minLength": 1,
          "maxLength": 29
        },
        "description": {
          "id": "/info/description",
          "type": "string",
          "minLength": 1,
          "maxLength": 199
        },
        "version": {
          "id": "/info/version",
          "type": "string",
          "minLength": 1,
          "maxLength": 63
        },
        "author-name": {
          "id": "/info/author-name",
          "type": "string",
          "minLength": 1,
          "maxLength": 199
        }
      }
    }
  }
}
```

```
"author-link": {
  "id": "/info/author-link",
  "type": "string",
  "minLength": 1,
  "maxLength": 199
}
},
"app": {
  "id": "/app",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "apptype": {
      "id": "/app/apptype",
      "type": "string",
      "minLength": 1,
      "maxLength": 63,
      "enum": [
        "lxc"
      ]
    },
    "cpuarch": {
      "id": "/app/cpuarch",
      "type": "string",
      "minLength": 1,
      "maxLength": 63,
      "enum": [
        "x86_64"
      ]
    }
  },
  "resources": {
    "id": "/app/resources",
    "type": "object",
    "additionalProperties": false,
    "properties": {
      "cpu": {
        "id": "/app/resources/cpu",
        "type": "integer",
        "multipleOf": 1,
        "maximum": 100,
        "minimum": 1
      },
      "memory": {
        "id": "/app/resources/memory",
        "type": "integer",
        "multipleOf": 1024,
        "minimum": 1024
      },
      "disk": {
        "id": "/app/resources/disk",
        "type": "array",
        "minItems": 1,
        "maxItems": 1,
        "uniqueItems": true,
        "items": {
          "id": "/app/resources/disk/0",
          "type": "object",
          "additionalProperties": false,
          "properties": {
            "target-dir": {
              "id": "/app/resources/disk/0/target-dir",
              "type": "string",
              "minLength": 1,

```

```

        "maxLength": 1,
        "enum": [
          "/"
        ]
      },
      "file": {
        "id": "/app/resources/disk/0/file",
        "type": "string",
        "minLength": 1,
        "maxLength": 63
      },
      "capacity": {
        "id": "/app/resources/disk/0/capacity",
        "type": "integer",
        "multipleOf": 1,
        "minimum": 1
      }
    }
  },
  "required": [
    "memory",
    "disk"
  ]
},
"required": [
  "apptype",
  "cpuarch",
  "resources"
]
},
"required": [
  "app"
]
}

```

show guestshell コマンド

show guestshell detail コマンドの出力には、ゲストシェルがインポートされたか、OVA からインストールされたかを示す情報が含まれます。

rootfsをインポートした後の **show guestshell detail** コマンドの例。

```

switch# show guestshell detail
Virtual service guestshell+ detail
State                : Activated
Package information
Name                  : rootfs_puppet
Path                  : usb2:/rootfs_puppet
Application
Name                  : GuestShell
Installed version    : 2.3(0.0)
Description           : Exported GuestShell: 20170613T173648Z
Signing
Key type              : Unsigned
Method                : Unknown
Licensing
Name                  : None

```

Version : None

仮想サービスと Guest Shell 情報の検証

次のコマンドを使用して、仮想サービスとゲストシェルの情報を検証できます。

コマンド	説明
<pre> show virtual-service global switch# show virtual-service global Virtual Service Global State and Virtualization Limits: Infrastructure version : 1.11 Total virtual services installed : 1 Total virtual services activated : 1 Machine types supported : LXC Machine types disabled : KVM Maximum VCPUs per virtual service : 1 Resource virtualization limits: Name Quota Committed Available ----- system CPU (%) 20 1 19 memory (MB) 3840 256 3584 bootflash (MB) 8192 200 7992 switch# </pre>	<p>仮想サービスのグローバル状態と制限を表示します。</p>
<pre> show virtual-service list switch# show virtual-service list * Virtual Service List: Name Status Package Name ----- guestshell+ Activated guestshell.ova chef Installed chef-0.8.1-n9000-spa-k9.ova </pre>	<p>仮想サービスの概要、仮想サービスのステータス、およびインストールされているソフトウェアパッケージを表示します。</p>

コマンド	説明
<pre> show guestshell detail switch# show guestshell detail Virtual service guestshell+ detail State : Activated Package information Name : guestshell.ova Path : /isan/bin/guestshell.ova Application Name : GuestShell Installed version : 2.2(0.2) Description : Cisco Systems Guest Shell Signing Key type : Cisco key Method : SHA-1 Licensing Name : None Version : None Resource reservation Disk : 400 MB Memory : 256 MB CPU : 1% system CPU Attached devices Type Name Alias ----- Disk _rootfs Disk /cisco/core Serial/shell Serial/aux Serial/Syslog serial2 Serial/Trace serial3 </pre>	<p>guestshell パッケージに関する詳細（バージョン、署名リソース、デバイスなど）を表示します。</p>

ゲスト シェルからのアプリケーションの永続的な起動

アプリケーションには、`/usr/lib/systemd/system/application_name.service` にインストールされる `systemd` / `systemctl` サービス ファイルが必要です。このサービス ファイルは、次の一般的なフォーマットにする必要があります。

```

[Unit]
Description=Put a short description of your application here

[Service]
ExecStart=Put the command to start your application here
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target

```




- (注) 特定のユーザーとして `systemd` を実行するには、サービスの [サービス (Service)] セクションに `User=<username>` を追加します。

Guest Shell からアプリケーションを永続的に起動する手順

手順

- ステップ 1** 上記で作成したアプリケーション サービス ファイルを `/usr/lib/systemd/system/application_name` にインストールします。サービス
- ステップ 2** `systemctl start application_name` でアプリケーションを開始します
- ステップ 3** アプリケーションが `systemctl status -l application_name` で実行されていることを確認します
- ステップ 4** `systemctl enable application_name` でリロード時にアプリケーションを再起動できるようにします
- ステップ 5** アプリケーションが `systemctl status -l application_name` で実行されていることを確認します

ゲストシェルでのサンプルアプリケーション

次の例は、ゲストシェルのアプリケーションを示しています。

```
root@guestshell guestshell]# cat /etc/init.d/hello.sh
#!/bin/bash

OUTPUTFILE=/tmp/hello

rm -f $OUTPUTFILE
while true
do
    echo $(date) >> $OUTPUTFILE
    echo 'Hello World' >> $OUTPUTFILE
    sleep 10
done
[root@guestshell guestshell]#
[root@guestshell guestshell]#
[root@guestshell system]# cat /usr/lib/systemd/system/hello.service
[Unit]
Description=Trivial "hello world" example daemon

[Service]
ExecStart=/etc/init.d/hello.sh &
Restart=always
RestartSec=10s
```

```
[Install]
WantedBy=multi-user.target
[root@guestshell system]#
[root@guestshell system]# systemctl start hello
[root@guestshell system]# systemctl enable hello
[root@guestshell system]# systemctl status -l hello
hello.service - Trivial "hello world" example daemon
   Loaded: loaded (/usr/lib/systemd/system/hello.service; enabled)
   Active: active (running) since Sun 2015-09-27 18:31:51 UTC; 10s ago
 Main PID: 355 (hello.sh)
   CGroup: /system.slice/hello.service
           ##355 /bin/bash /etc/init.d/hello.sh &
           ##367 sleep 10

Sep 27 18:31:51 guestshell hello.sh[355]: Executing: /etc/init.d/hello.sh &
[root@guestshell system]#
[root@guestshell guestshell]# exit
exit
[guestshell@guestshell ~]$ exit
logout
switch# reload
This command will reboot the system. (y/n)? [n] y
```

リロード後

```
[root@guestshell guestshell]# ps -ef | grep hello
root      20      1  0 18:37 ?        00:00:00 /bin/bash /etc/init.d/hello.sh &
root     123    108  0 18:38 pts/4    00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
[root@guestshell guestshell]# cat /tmp/hello
Sun Sep 27 18:38:03 UTC 2015
Hello World
Sun Sep 27 18:38:13 UTC 2015
Hello World
Sun Sep 27 18:38:23 UTC 2015
Hello World
Sun Sep 27 18:38:33 UTC 2015
Hello World
Sun Sep 27 18:38:43 UTC 2015
Hello World
[root@guestshell guestshell]#
```

systemd / systemctl で実行すると、アプリケーションが停止した場合（または強制終了した場合）、アプリケーションは自動的に再起動されます。プロセス識別子はもともと 226 です。アプリケーションを強制終了すると、プロセス識別子 257 で自動的に再起動されます。

```
[root@guestshell guestshell]# ps -ef | grep hello
root      226      1  0 19:02 ?        00:00:00 /bin/bash /etc/init.d/hello.sh &
root     254    116  0 19:03 pts/4    00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
[root@guestshell guestshell]# kill -9 226
[root@guestshell guestshell]#
[root@guestshell guestshell]# ps -ef | grep hello
root      257      1  0 19:03 ?        00:00:00 /bin/bash /etc/init.d/hello.sh &
root     264    116  0 19:03 pts/4    00:00:00 grep --color=auto hello
[root@guestshell guestshell]#
```

翻訳について

このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。