Guest Shell

- About the Guest Shell, on page 1
- Guidelines and Limitations for Guestshell, on page 2
- Accessing the Guest Shell, on page 7
- Resources Used for the Guest Shell, on page 7
- Capabilities in the Guestshell, on page 8
- Security Posture for Virtual ServicesGuest Shell, on page 16
- Guest File System Access Restrictions , on page 19
- Managing the Guest Shell, on page 19
- Verifying Virtual Service and Guest Shell Information, on page 30
- Persistently Starting Your Application From the Guest Shell, on page 32
- Procedure for Persistently Starting Your Application from the Guest Shell, on page 33
- An Example Application in the Guest Shell, on page 33
- Troubleshooting Guest Shell Issues, on page 34

About the Guest Shell

In addition to the NX-OS CLI and Bash access on the underlying Linux environment, switches support access to a decoupled execution space running within a Linux Container (LXC) called the “Guest Shell”.

From within the Guest Shell the network-admin has the following capabilities:

- Access to the network over Linux network interfaces.
- Access to the switch's bootflash.
- Access to the switch's volatile tmpfs.
- Access to the switch's CLI.
- Access to the switch's host file system.
- Access to Cisco NX-API REST.
- The ability to install and run python scripts.
- The ability to install and run 32-bit and 64-bit Linux applications.

Decoupling the execution space from the native host system allows customization of the Linux environment to suit the needs of the applications without impacting the host system or applications running in other Linux Containers.

On NX-OS devices, Linux Containers are installed and managed with the virtual-service commands. The Guest Shell will appear in the virtual-service show command output.

Note By default, the Guest Shell occupies approximately 5 MB of RAM and 200 MB of bootflash when enabled. Beginning with Cisco NX-OS Release 7.0(3)I2(1) the Guest Shell occupies approximately 35 MB of RAM. Use the **guestshell destroy** command to reclaim resources if the Guest Shell is not used.

Note By default, the Guest Shell occupies approximately 35 MB of RAM and 350 MB of bootflash when enabled. Use the **guestshell destroy** command to reclaim resources if the Guest Shell is not used.

Note Beginning with Cisco NX-OS 7.0(3)F3(1), the Guest Shell is supported on the Cisco Nexus 9508 switch.

Guidelines and Limitations for Guestshell

Common Guidelines Across All Releases

Important If you have performed custom work inside your installation of the Guestshell, save your changes to the bootflash, off-box storage, or elsewhere outside the Guestshell root file system before performing a `guestshell upgrade`.

The `guestshell upgrade` command essentially performs a `guestshell destroy` and `guestshell enable` in succession.

- Guest Shell is not supported on 3500 models with 4GB of memory (3524, 3548, 3524-X, 3548-X). It is supported on the platforms with higher memory, such as -XL.
- If you are running a third-party DHCPD server in Guestshell, there might be issues with offers reaching the client if used along with SVI. A possible workaround is to use broadcast responses.
- Use the `run guestshell` CLI command to access the Guestshell on the switch: The `run guestshell` command parallels the `run bash` command that is used to access the host shell. This command allows you to access the Guestshell and get a Bash prompt or run a command within the context of the Guestshell. The command uses password-less SSH to an available port on the localhost in the default network namespace.
- The `sshd` utility can secure the pre-configured SSH access into the Guestshell by listening on `localhost` to avoid connection attempts from outside the network. The `sshd` has the following features:
    - It is configured for key-based authentication without fallback to passwords.

- Only `root` can read keys use to access the Guestshell after Guestshell restarts.
- Only `root` can read the file that contains the key on the host to prevent a nonprivileged user with host Bash access from being able to use the key to connect to the Guestshell. Network-admin users may start another instance of sshd in the Guestshell to allow remote access directly into the Guestshell, but any user that logs into the Guestshell is also given network-admin privilege.

Note Introduced in Guestshell 2.2 (0.2), the key file is readable for whom the user account was created for.

In addition, the Guestshell accounts are not automatically removed, and must be removed by the network administrator when no longer needed.

Guestshell installations before 2.2 (0.2) will not dynamically create individual user accounts.

- Installing the Cisco NX-OS software release on a fresh out-of-the-box switch will automatically enable the Guestshell. Subsequent upgrades to the switch software will not automatically upgrade Guestshell.
- Guestshell releases increment the major number when distributions or distribution versions change.
- Guestshell for NX-OS can access front-panel ports as first-class Linux interfaces.
- Guestshell for NX-OS can access Command shell through dohost using local Unix socket to NX-API.
  1. Guestshell for NX-OS: Access to NX-API socket is allowed only for root/admin user privilege from 9.3(8) and later.
  2. Guestshell for NX-OS: Access to NX-OS filesystem only as root/admin user in 9.3(8) and later.
- Guestshell releases increment the minor number when CVEs have been addressed. The Guestshell updates CVEs only when CentOS makes them publicly available.
- Cisco recommends using **yum update** to pick up third-party security vulnerability fixes directly from the CentOS repository. This provides the flexibility of getting updates as, and when, available without needing to wait for a Cisco NX-OS software update.

  Alternatively, using the **guestshell update** command would replace the existing Guestshell rootfs. Any customizations and software package installations would then need to be performed again within the context of this new Guestshell rootfs.

CentOS end of life and impact on Guestshell

Guestshell is an **LXC container based on CentOS environment.** As per updates in the open source community, CentOS 8 Project is reaching end of support by December 2021. The CentOS 7 project is to continue through and is targeted to reach end of support by June 2024. Due to this long term support for CentOS 7, the latest Cisco NX-OS software 10.2.x is packaged with Guestshell 2.11 (CentOS 7 based). This replaces Guestshell 3.0 (CentOS 8) which is the default environment in 10.1.x release.

Guestshell 2.11

Beginning with Cisco NX-OS release 10.2(1), CentOS 7 is re-introduced as the default Guestshell environment. See section "*CentOS End of Life*" for a detailed explanation on the reasons.

Guestshell 2.11 comes with python2 and python3.6 support. The functionality between Guestshell 2.11 and Guestshell 3.0 remains the same.

Note The rootfs size of Guestshell 2.11 has increased to approximately 200 MB.

Upgrading from Guestshell 1.0 to Guestshell 2.x

Guestshell 2.x is based on a CentOS 7 root file system. If you have an off-box repository of `.conf` files or utilities that pulled the content down into Guestshell 1.0, you must repeat the same deployment steps in Guestshell 2.x. Your deployment script may need to be adjusted to account for the CentOS 7 differences.

Guestshell 2.x

The Cisco NX-OS automatically installs and enables the Guestshell by default on systems with sufficient resources. However, if the device is reloaded with a Cisco NX-OS image that does not provide Guestshell support, the installer will automatically remove the existing Guestshell and issue a `%VMAN-2-INVALID_PACKAGE`.

Note Systems with 4 GB of RAM will not enable Guestshell by default. Use the **guestshell enable** command to install and enable Guestshell.

The **install all** command validates the compatibility between the current Cisco NX-OS image against the target Cisco NX-OS image.

The following is an example output from installing an incompatible image:

```
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: Guestshell not supported, do 'guestshell destroy' to remove
it and then retry ISSU
Pre-upgrade check failed. Return code 0x42DD0006 ((null)).
switch#
```

Note As a best practice, remove the Guestshell with the **guestshell destroy** command before reloading an older Cisco NX-OS image that does not support the Guestshell.

Pre-Configured SSHD Service

The Guestshell starts an OpenSSH server upon boot up. The server listens on a randomly generated port on the localhost IP address interface 127.0.0.1 only. This provides the password-less connectivity into the Guestshell from the NX-OS virtual-shell when the guestshell keyword is entered. If this server is killed or its configuration (residing in `/etc/ssh/sshd_config-cisco`) is altered, access to the Guestshell from the NX-OS CLI might not work.

The following steps instantiate an OpenSSh server within the Guestshell as root:

1. Determine which network namespace or VRF you want to establish your SSH connections through.
2. Determine the port that you want OpenSSH to listen on. Use the NX-OS command **show socket connection** to view ports already in use.

Note The Guestshell sshd service for password-less access uses a randomized port starting at 17680 through 49150. To avoid port conflict, choose a port outside this range.

The following steps start the OpenSSH server. The examples start the OpenSSH server for management netns on IP address 10.122.84.34:2222:

1. Create the following files: `/usr/lib/systemd/systm/sshd-mgmt.service` and `/etc/ssh/sshd-mgmt_config`. The files should have the following configurations:

   ```
   -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. Copy the Unit and Service contents from the `/usr/lib/systemd/system/ssh.service` file to `sshd-mgmt.service`.

3. Edit the `sshd-mgmt.service` file to match the following:

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

   [Service]
   EnvironmentFile=/etc/sysconfig/sshd
   ```

```
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. Copy the contents of `/etc/ssh/sshd-config` to `/etc/ssh/sshd-mgmt_config`. Modify the ListenAddress IP and port as necessary.

   ```
   Port 2222
   ListenAddress 10.122.84.34
   ```

5. Start the systemctl daemon using the following commands:

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

6. (Optional) Check the configuration.

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

7. SSH into Guestshell:

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

8. Save the configuration across multiple Guestshell or switch reboots.

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

9. For passwordless SSH/SCP and remote execution, generate the public and private keys for the user ID you want to user for SSH/SCP using the **ssh-keygen -t dsa** command.

   The key is then stored in the `id_rsa` and `id_rsa.pub` files in the `/.ssh` directory:

   ```
   [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. Copy the public key into the machine you want to SSH into and fix permissions:

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

11. SSH or SCP into the remote switch without a password:

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

Localtime

The Guestshell shares `/etc/localtime` with the host system.

Note If you do not want to share the same localtime with the host, this symlink can be broken and a Guestshell specific `/etc/localtime` can be created.

```
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
```

Accessing the Guest Shell

In Cisco NX-OS, only network-admin users can access the Guest Shell by default. It is automatically enabled in the system and can be accessed using the **run guestshell** command. Consistent with the **run bash** command, these commands can be issued within the Guest Shell with the **run guestshell** *command* form of the NX-OS CLI command.

Note The Guest Shell is automatically enabled on systems with more than 4 GB of RAM.

```
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
```

Note The Guest Shell starting in 2.2(0.2) will dynamically create user accounts with the same as the user logged into switch. However, all other information is NOT shared between the switch and the Guest Shell user accounts.

In addition, the Guest Shell accounts are not automatically removed, and must be removed by the network administrator when no longer needed.

Resources Used for the Guest Shell

By default, the resources for the Guest Shell have a small impact on resources available for normal switch operations. If the network-admin requires additional resources for the Guest Shell, the **guestshell resize** {*cpu* | *memory* | *rootfs*} command changes these limits.

Resource	Default	Minimum/Maximum
CPU	1%	1/20%
Memory	400 MB	256/3840 MB
Storage	200 MB	200/2000 MB

The CPU limit is the percentage of the system compute capacity that tasks running within the Guest Shell are given when there is contention with other compute loads in the system. When there is no contention for CPU resources, the tasks within the Guest Shell are not limited.

Note A Guest Shell reboot is required after changing the resource allocations. This can be accomplished with the **guestshell reboot** command.

Capabilities in the Guestshell

The Guestshell has a number of utilities and capabilities available by default.

The Guestshell is populated with CentOS 7 Linux which provides the ability to dnf install software packages built for this distribution. The Guestshell is pre-populated with many of the common tools that would naturally be expected on a networking device including **net-tools**, **iproute**, **tcpdump** and OpenSSH. For Guestshell 2.x, python 2.7.5 is included by default as is the PIP for installing additional python packages. In Guestshell 2.11, by default, python 3.6 is also included.

By default the Guestshell is a 64-bit execution space. If 32-bit support is needed, the glibc.i686 package can be dnf installed.

The Guestshell has access to the Linux network interfaces used to represent the management and data ports of the switch. Typical Linux methods and utilities like **ifconfig** and **ethtool** can be used to collect counters. When an interface is placed into a VRF in the NX-OS CLI, the Linux network interface is placed into a network namespace for that VRF. The name spaces can be seen at `/var/run/netns` and the **ip netns** utility can be used to run in the context of different namespaces. A couple of utilities, **chvrf** and **vrfinfo**, are provided as a convenience for running in a different namespace and getting information about which namespace/vrf a process is running in.

systemd is used to manage services in CentOS 8 environments, including the Guestshell.

NX-OS CLI in the Guest Shell

The Guest Shell provides an application to allow the user to issue NX-OS commands from the Guest Shell environment to the host network element. The **dohost** application accepts any valid NX-OS configuration or exec commands and issues them to the host network element.

When invoking the **dohost** command each NX-OS command may be in single or double quotes:

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

The NX-OS CLI can be chained together:

```
[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 ~]$
```

The NX-OS CLI can also be chained together using the NX-OS style command chaining technique by adding a semicolon between each command. (A space on either side of the semicolon is required.):

```
[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 ~]$
```

Note For release 7.0(3)I5(2) usingStarting with Guest Shell 2.2 (0.2), commands issued on the host through the **dohost** command are run with privileges based on the effective role of the Guest Shell user.

Prior versions of Guest Shell will run command with network-admin level privileges.

The **dohost** command fails when the number of UDS connections to NX-API are at the maximum allowed.

Network Access in Guest Shell

The NX-OS switch ports are represented in the Guest Shell as Linux network interfaces. Typical Linux methods like view stats in /proc/net/dev, through ifconfig or ethtool are all supported:

The Guest Shell has a number of typical network utilities included by default and they can be used on different VRFs using the **chvrf** *vrf command* command.

```
[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
```

Within the Guest Shell, the networking state can be monitored, but may not be changed. To change networking state, use the NX-OS CLI or the appropriate Linux utilities in the host bash shell.

The **tcpdump** command is packaged with the Guest Shell to allow packet tracing of punted traffic on the management or switch ports.

The **sudo ip netns exec management ping** utility is a common method for running a command in the context of a specified network namespace. This can be done within the Guest Shell:

```
[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
```

The chvrf utility is provided as a convenience:

```
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
```

Note Commands that are run without the **chvrf** command are run in the current VRF/network namespace.

For example, to ping IP address 10.0.0.1 over the management VRF, the command is “**chvrf** *management ping 10.0.0.1*”. Other utilities such as **scp** or **ssh** would be similar.

Example:

```
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]$
```

To obtain a list of VRFs on the system, use the **show vrf** command natively from NX-OS or through the **dohost** command:

Example:

```
[guestshell@guestshell bootflash]$ dohost 'sh vrf'
VRF-Name   VRF-ID  State     Reason
default    1       Up        --
management 2       Up        --
red        6       Up        --
```

Within the Guest Shell, the network namespaces associated with the VRFs are what is actually used. It can be more convenient to just see which network namespaces are present:

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

To resolve domain names from within the Guest Shell, the resolver needs to be configured. Edit the /etc/resolv.conf file in the Guest Shell to include a DNS nameserver and domain as appropriate for the network.

Example:

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

The nameserver and domain information should match what is configured through the NX-OS configuration.

Example:

```
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
```

If the switch is in a network that uses an HTTP proxy server, the **http_proxy** and **https_proxy** environment variables must be set up within the Guest Shell also.

Example:

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

These environment variables should be set in the .bashrc file or in an appropriate script to ensure that they are persistent.

Access to Bootflash in Guest Shell

Network administrators can manage files with Linux commands and utilities in addition to using NX-OS CLI commands. By mounting the system bootflash at /bootflash in the Guest Shell environment, the network-admin can operate on these files with Linux commands.

Example:

```
find . –name “foo.txt”
rm “/bootflash/junk/foo.txt”
```

Note While the name of the user within the Guest Shell is the same as when on the host, the Guest Shell is in a separate user namespace, and the uid does not match that of the user on the host. The file permissions for group and others will control the type of access the Guest Shell user has on the file.

Python in Guest Shell

Python can be used interactively or python scripts can be run in the Guest Shell.

Example:

```
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:~$
```

The pip python package manager is included in the Guest Shell to allow the network-admin to install new python packages.

Example:

```
[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]#
```

Note You must enter the **sudo su** command before entering the **pip install** command.

Python 3 in Guest Shell versions up to 2.10 (CentOS 7)

Guest Shell 2.X provides a CentOS 7.1 environment, which does not have Python 3 installed by default. There are multiple methods of installing Python 3 on CentOS 7.1, such as using third-party repositories or building from source. Another option is using the Red Hat Software Collections, which supports installing multiple versions of Python within the same system.

To install the Red Hat Software Collections (SCL) tool:

1. Install the scl-utils package.
2. Enable the CentOS SCL repository and install one of its provided Python 3 RPMs.

```
[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-sclo-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!
```

Using SCL, it is possible to create an interactive bash session with Python 3's environment variables automatically setup.

Note The root user is not needed to use the SCL Python installation.

```
[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.
>>>
```

The Python SCL installation also provides the pip utility.

```
[admin@guestshell ~]$ pip3 install requests --user
Collecting requests
  Downloading
https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a00aba99ae0f828be89d72b2190f27c11d4b7fb/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/cd551d81dbe15200be1cf41cd03869a46fe7226e7450af7a6545bfc474c9/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/a9/01ffebfb562e4274b6487b4bb1ddec7ca55ec7510b22e4c51f14098443b8/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/df50cac98ea0d5b006c55a399c3bf1db9da7b5a24de7890bc9cfd5dd9e99/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/e8/74/6e4f91745020f967d09332bb2b8b9b10090957334692eb88ea4afe91b77f/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]>
```

The default Python 2 installation can be used alongside the SCL Python installation.

```
[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 makes it possible to install multiple versions of the same RPM on a system. In this case, it is possible to install Python 3.5 in addition to Python 3.6.

```
[admin@guestshell ~]$ sudo yum install -y rh-python35 | tail
Dependency Installed:
  rh-python35-python.x86_64 0:3.5.1-13.el7
  rh-python35-python-devel.x86_64 0:3.5.1-13.el7
  rh-python35-python-libs.x86_64 0:3.5.1-13.el7
  rh-python35-python-pip.noarch 0:7.1.0-2.el7
  rh-python35-python-setuptools.noarch 0:18.0.1-2.el7
  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.
>>>
```

Note Creating new interactive bash sessions when multiple Python versions are installed in SCL can cause an issue where the libpython shared object file cannot be loaded. There is a workaround where you can use the **source scl_source enable** *python-installation* command to properly set up the environment in the current bash session.

The default Guest Shell storage capacity is not sufficient to install Python 3. Use the **guestshell resize rootfs** *size-in-MB* command to increase the size of the file system. Typically, setting the rootfs size to 550 MB is sufficient.

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.corespace.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 roofs** *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.

Security Posture for Virtual ServicesGuest Shell

Use of the Guest Shell and virtual services in switches are only two of the many ways that the network-admin can manage or extend the functionality of the system. These options are geared toward providing an execution environment that is decoupled from the native host context. This separation allows the introduction of software into the system that may not be compatible with the native execution environment. It also allows the software to run in an environment that does not interfere with the behavior, performance, or scale of the system.

Use of the Guest Shell in switches is just one of the many ways the network admin can manage or extend the functionality of the system. The Guest Shell is intended to provide an execution environment that is decoupled from the native host context. This separation allows the introduction of software into the system that may not be compatible with the native execution environment. It also allows the software to run in an environment that does not interfere with the behavior, performance, or scale of the system.

Kernel Vulnerability Patches

Cisco responds to pertinent Common Vulnerabilities and Exposures (CVEs) with platform updates that address known vulnerabilities.

ASLR and X-Space Support

Cisco 9000 NX-OS supports the use of Address Space Layout Randomization (ASLR) and Executable Space Protection (X-Space) for runtime defense. The software in Cisco-signed packages make use of this capability. If other software is installed on the system, it is recommended that it be built using a host OS and development toolchain that supports these technologies. Doing so reduces the potential attack surface that the software presents to potential intruders.

Namespace Isolation

The Guest Shell environment runs within a Linux container that makes use of various namespaces to decouple the Guest Shell execution space from that of the host. Starting in the NX-OS 9.2(1) release, the Guest Shell is run in a separate user namespace, which helps protect the integrity of the host system, as processes running as root within the Guest Shell are not root of the host. These processes appear to be running as uid 0 within the Guest Shell due to uid mapping, but the kernel knows the real uid of these processes and evaluates the POSIX capabilities within the appropriate user namespace.

When a user enters the Guest Shell from the host, a user of the same name is created within the Guest Shell. While the names match, the uid of the user within the Guest Shell is not the same as the uid on the host. To still allow users within the Guest Shell to access files on shared media (for example, /bootflash or /volatile), the common NX-OS gids used on the host (for example, network-admin or network-operator) are mapped into the Guest Shell such that the values are the same and the Guest Shell instance of the user is associated with the appropriate groups based on group membership on the host.

As an example, consider user bob. On the host, bob has the following uid and gid membership:

```
bash-4.3$ id
 uid=2004(bob) gid=503(network-admin) groups=503(network-admin),504(network-operator)
```

When user bob is in the Guest Shell, the group membership from the host is set up in the Guest Shell:

```
[bob@guestshell ~]$ id
 uid=1002(bob) gid=503(network-admin)
groups=503(network-admin),504(network-operator),10(wheel)
```

Files created by user bob in the host Bash shell and the Guest Shell have different owner ids. The example output below shows that the file created from within the Guest Shell has owner id 12002, instead of 1002 as shown in the example output above. This is due to the command being issued from the host Bash shell and the id space for the Guest Shell starting at id 11000. The group id of the file is network-admin, which is 503 in both environments.

```
bash-4.3$ ls -ln /bootflash/bob_*
-rw-rw-r-- 1 12002 503 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 2004 503 4 Jun 22 15:47 /bootflash/bob_host


bash-4.3$ ls -l /bootflash/bob_*
-rw-rw-r-- 1 12002 network-admin 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 bob network-admin 4 Jun 22 15:47 /bootflash/bob_host
```

The user is allowed to access the file due to the file permission settings for the network-admin group, and the fact that bob is a member of network-admin in both the host and Guest Shell.

Inside the Guest Shell environment, the example output below shows that the owner id for the file created by bob from the host is 65534. This indicates the actual id is in a range that is outside range of ids mapped into the user namespace. Any unmapped id will be shown as this value.

```
[bob@guestshell ~]$ ls -ln /bootflash/bob_*
-rw-rw-r-- 1 1002 503 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 65534 503 4 Jun 22 15:47 /bootflash/bob_host


[bob@guestshell ~]$ ls -l /bootflash/bob_*
-rw-rw-r-- 1 bob network-admin 4 Jun 22 15:47 /bootflash/bob_guestshell
-rw-rw-r-- 1 65534 network-admin 4 Jun 22 15:47 /bootflash/bob_host
```

Root-User Restrictions

As a best practice for developing secure code, it is recommend running applications with the least privilege needed to accomplish the assigned task. To help prevent unintended accesses, software added into the Guest Shell should follow this best practice.

All processes within a virtual servicethe Guest Shell are subject to restrictions imposed by reduced Linux capabilities. If your application must perform operations that require root privileges, restrict the use of the root account to the smallest set of operations that absolutely requires root access, and impose other controls such as a hard limit on the amount of time that the application can run in that mode.

The set of Linux capabilities that are dropped for root within virtual servicesthe Guest Shell follow:

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	

- cap_audit_control
- cap_audit_write
- cap_mac_admin
- cap_mac_override
- cap_mknod
- cap_net_broadcast
- cap_sys_boot
- cap_syslog
- cap_sys_module
- cap_sys_nice
- cap_sys_pacct
- cap_sys_ptrace
- cap_sys_rawio
- cap_sys_resource
- cap_sys_time
- cap_wake_alarm

As root within a virtual service, bind mounts may be used as well as tmpfs and ramfs mounts. Other mounts are prevented.

While the net_admin capability is not dropped, user namespace and the host ownership of the network namespaces prevents the Guest Shell user from modifying the interface state. As root within the Guest Shell, bind mounts may be used as well as tmpfs and ramfs mounts. Other mounts are prevented.

Resource Management

A Denial-of-Service (DoS) attack attempts to make a machine or network resource unavailable to its intended users. Misbehaving or malicious application code can cause DoS as the result of over-consumption of connection bandwidth, disk space, memory, and other resources. The host provides resource-management features that ensure fair allocation of resources among all virtual servicesbetween Guest Shell and services on the host.

Guest File System Access Restrictions

To preserve the integrity of the files within the virtual services, the file systems of the virtual services are not accessible from the NX-OS CLI. If a given virtual-service allows files to be modified, it needs to provide an alternate means by which this can be done (i.e. **yum install**, **scp**, **ftp**, etc).

To preserve the integrity of the files within the Guest Shell, the file systems of the Guest Shell are not accessible from the NX-OS CLI.

`bootflash:` and `volatile:` of the host are mounted as `/bootflash` and `/volatile` within the Guest Shell. A network-admin can access files on this media using the NX-OS exec commands from the host or using Linux commands from within the Guest Shell.

Managing the Guest Shell

The following are commands to manage the Guest Shell:

Table 1: Guest Shell CLI Commands

Commands	Description
guestshell enable {**package** [*guest shell OVA file* | *rootfs-file-URI*]}	• When *guest shell OVA file* is specified: Installs and activates the Guest Shell using the OVA that is embedded in the system image. Installs and activates the Guest Shell using the specified software package (OVA file) or the embedded package from the system image (when no package is specified). Initially, Guest Shell packages are only available by being embedded in the system image. When the Guest Shell is already installed, this command enables the installed Guest Shell. Typically this is used after a **guestshell disable** command. • When *rootfs-file-URI* is specified: Imports a Guest Shell **rootfs** when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.
guestshell export rootfs package *destination-file-URI*	Exports a Guest Shell **rootfs** file to a local URI (bootflash, USB1, etc.). (7.0(3)I7(1) and later releases)
guestshell disable	Shuts down and disables the Guest Shell.

Commands	Description
guestshell upgrade {**package** [*guest shell OVA file* | *rootfs-file-URI*]}	• When *guest shell OVA file* is specified: Deactivates and upgrades the Guest Shell using the specified software package (OVA file) or the embedded package from the system image (if no package is specified). Initially Guest Shell packages are only available by being embedded in the system image. The current rootfs for the Guest Shell is replaced with the rootfs in the software package. The Guest Shell does not make use of secondary filesystems that persist across an upgrade. Without persistent secondary filesystems, a **guestshell destroy** command followed by a **guestshell enable** command could also be used to replace the rootfs. When an upgrade is successful, the Guest Shell is activated. You are prompted for a confirmation prior to carrying out the upgrade command. • When *rootfs-file-URI* is specified: Imports a Guest Shell **rootfs** file when the Guest Shell is already installed. This command removes the existing Guest Shell and installs the specified package.
guestshell reboot	Deactivates the Guest Shell and then reactivates it. You are prompted for a confirmation prior to carrying out the reboot command. **Note** This is the equivalent of a **guestshell disable** command followed by a **guestshell enable** command in exec mode. This is useful when processes inside the Guest Shell have been stopped and need to be restarted. The **run guestshell** command relies on `sshd` running in the Guest Shell. If the command does not work, the `sshd` process may have been inadvertently stopped. Performing a reboot of the Guest Shell from the NX-OS CLI allows it to restart and restore the command.

Commands	Description
guestshell destroy	Deactivates and uninstalls the Guest Shell. All resources associated with the Guest Shell are returned to the system. The **show virtual-service global** command indicates when these resources become available. Issuing this command results in a prompt for a confirmation prior to carrying out the destroy command.
guestshell **run guestshell**	Connects to the Guest Shell that is already running with a shell prompt. No username/password is required.
guestshell run *command* **run guestshell** *command*	Executes a Linux/UNIX command within the context of the Guest Shell environment. After execution of the command you are returned to the switch prompt.
guestshell resize [**cpu** | **memory** | **rootfs**]	Changes the allotted resources available for the Guest Shell. The changes take effect the next time the Guest Shell is enabled or rebooted. **Note** Resize values are cleared when the **guestshell destroy** command is used.
guestshell sync	On systems that have active and standby supervisors, this command synchronizes the Guest Shell contents from the active supervisor to the standby supervisor. The network-admin issues this command when the Guest Shell rootfs has been set up to a point that they would want the same rootfs used on the standby supervisor when it becomes the active supervisor. If this command is not used, the Guest Shell is freshly installed when the standby supervisor transitions to an active role using the Guest Shell package available on that supervisor.
virtual-service reset force	In the event that the guestshell or virtual-services cannot be managed, even after a system reload, the reset command is used to force the removal of the Guest Shell and all virtual-services. The system needs to be reloaded for the cleanup to happen. No Guest Shell or additional virtual-services can be installed or enabled after issuing this command until after the system has been reloaded. You are prompted for a confirmation prior to initiating the reset.

Note Administrative privileges are necessary to enable/disable and to gain access to the Guest Shell environment.

Note The Guest Shell is implemented as a Linux container (LXC) on the host system. On NX-OS devices, LXCs are installed and managed with the virtual-service commands. The Guest Shell appears in the virtual-service commands as a virtual service named `guestshell+`.

Note Virtual-service commands that do not pertain to the Guest Shell are being deprecated. These commands have been hidden in the NX-OS 9.2(1) release and will be removed in future releases.

The following exec keywords are being deprecated:

```
# virtual-service ?
connect  Request a virtual service shell
install  Add a virtual service to install database
uninstall  Remove a virtual service from the install database
upgrade  Upgrade a virtual service package to a different version


# show virtual-service ?
detail  Detailed information config)
```

The following config keywords are being deprecated:

```
(config) virtual-service ?
WORD  Virtual service name (Max Size 20)


(config-virt-serv)# ?
activate  Activate configured virtual service
description  Virtual service description
```

Disabling the Guest Shell

The **guestshell disable** command shuts down and disables the Guest Shell.

When the Guest Shell is disabled and the system is reloaded, the Guest Shell remains disabled.

Example:

```
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
```

Note The Guest Shell is reactivated with the **guestshell enable** command.

Destroying the Guest Shell

The **guestshell destroy** command uninstalls the Guest Shell and its artifacts. The command does not remove the Guest Shell OVA.

When the Guest Shell is destroyed and the system is reloaded, the Guest Shell remains destroyed.

```
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:
```

Note The Guest Shell can be re-enabled with the **guestshell enable** command.

Note If you do not want to use the Guest Shell, you can remove it with the **guestshell destroy** command. Once the Guest Shell has been removed, it remains removed for subsequent reloads. This means that when the Guest Shell container has been removed and the switch is reloaded, the Guest Shell container is not automatically started.

Enabling the Guest Shell

The **guestshell enable** command installs the Guest Shell from a Guest Shell software package. By default, the package embedded in the system image is used for the installation. The command is also used to reactivate the Guest Shell if it has been disabled.

When the Guest Shell is enabled and the system is reloaded, the Guest Shell remains enabled.

Example:

```
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
```

Enabling the Guest Shell in Base Boot Mode

Beginning in the NX-OS 9.2(1) release, you can choose to boot your system in *base boot mode*. When you boot your system in base boot mode, the Guest Shell is not started by default. In order to use the Guest Shell in this mode, you must activate the RPMs containing the virtualization infrastructure as well as the Guest Shell image. Once you have done this, the Guest Shell and virtual-service commands will be available.

If the RPM activation commands are run in this order:

1. `install activate guestshell`
2. `install activate virtualization`

The Guest Shell container will be activated automatically as it would have been if the system had been booted in full mode.

If the RPM activation commands are run in the reverse order:

1. `install activate virtualization`
2. `install activate guestshell`

Then the Guest Shell will not be enabled until you run the **guestshell enable** command.

Replicating the Guest Shell

Beginning with Cisco NX-OS release 7.0(3)I7(1), a Guest Shell **rootfs** that is customized on one switch can be deployed onto multiple switches.

The approach is to customize and then export the Guest Shell **rootfs** and store it on a file server. A POAP script can download (import) the Guest Shell **rootfs** to other switches and install the specific Guest Shell across many devices simultaneously.

Exporting Guest Shell rootfs

Use the **guestshell export rootfs package** *destination-file-URI* command to export a Guest Shell **rootfs**.

The *destination-file-URI* parameter is the name of the file that the Guest Shell **rootfs** is copied to. This file allows for local URI options (bootflash, USB1, etc.).

The **guestshell export rootfs package** command:

- Disables the Guest Shell (if already enabled).
- Creates a Guest Shell import YAML file and inserts it into the /cisco directory of the **rootfs** ext4 file.
- Copies the **rootfs** ext4 file to the target URI location.
- Re-enables the Guest Shell if it had been previously enabled.

Importing Guest Shell rootfs

When importing a Guest Shell **rootfs**, there are two situations to consider:

- Use the **guestshell enable package** *rootfs-file-URI* command to import a Guest Shell **rootfs** when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.
- Use the **guestshell upgrade package** *rootfs-file-URI* command to import a Guest Shell **rootfs** when the Guest Shell is already installed. This command removes the existing Guest Shell and installs the specified package.

The *rootfs-file-URI* parameter is the **rootfs** file stored on local storage (bootflash, USB, etc.).

When this command is executed with a file that is on bootflash, the file is moved to a storage pool on bootflash.

As a best practice, you should copy the file to the bootflash and validate the md5sum before using the **guestshell upgrade package** *rootfs-file-URI* command.

Note The **guestshell upgrade package** *rootfs-file-URI* command can be executed from within the Guest Shell.

Note The rootfs file is not a Cisco signed package, you must configure to allow unsigned packages before enabling as shown in the example:

```
(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.
```

Note To restore the embedded version of the rootfs:

- Use the **guestshell upgrade** command (without additional parameters) when the Guest Shell has already been installed.
- Use the **guestshell enable** command (without additional parameters) when the Guest Shell had been destroyed.

Note When running this command from within a Guest Shell, or outside a switch using NX-API, you must set **terminal dont-ask** to skip any prompts.

The **guestshell enable package** *rootfs-file-URI* command:

- Performs basic validation of the **rootfs** file.
- Moves the **rootfs** into the storage pool.
- Mounts the **rootfs** to extract the YAML file from the /cisco directory.
- Parses the YAML file to obtain VM definition (including resource requirements).
- Activates the Guest Shell.

Example workflow for **guestshell enable** :

```
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+'
```

Note Workflow for **guestshell upgrade** is preceded by the existing Guest Shell being destroyed.

Note Resize values are cleared when the **guestshell upgrade** command is used.

Importing YAML File

A YAML file that defines some user modifiable characteristics of the Guest Shell is automatically created as a part of the export operation. It is embedded into the Guest Shell **rootfs** in the /cisco directory. It is not a complete descriptor for the Guest Shell container. It only contains some of the parameters that are user modifiable.

Example of a Guest Shell import YAML file:

```
---
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
...
```

The YAML file is generated when the **guestshell export rootfs package** command is executed. The file captures the values of the currently running Guest Shell.

The info section contains non-operational data that is used to help identify the Guest Shell. Some of the information will be displayed in the output of the **show guestshell detail** command.

The description value is an encoding of the UTC time when the YAML file was created. The time string format is the same as DTSTAMP in RFC5545 (iCal).

The resources section describes the resources required for hosting the Guest Shell. The value "/" for the target-dir in the example identifies the disk as the **rootfs**.

Note If resized values were specified while the Guest Shell was destroyed, those values take precedence over the values in the import YAML file when the **guestshell enable package** command is used.

The cpuarch value indicates the CPU architecture that is expected for the container to run.

You can modify the YAML file (such as the description or increase the resource parameters, if appropriate) after the export operation is complete .

Cisco provides a python script that you can run to validate a modified YAML file with a JSON schema. It is not meant to be a complete test (for example, device-specific resource limits are not checked), but it is able to flag common errors. The python script with examples is located at Guest Shell Import Export. The following JSON file describes the schema for version 1.0 of the Guest Shell import YAML .

```
{
  "$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 Command

The output of the **show guestshell detail** command includes information that indicates whether the Guest Shell was imported or was installed from an OVA.

Example of the **show guestshell detail** command after importing **rootfs**.

```
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
```

Verifying Virtual Service and Guest Shell Information

You can verify virtual service and Guest Shell information with the following commands:

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>

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#
```

</td><td>Displays the global state and limits for virtual services.</td></tr>
<tr><td>

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
```

</td><td>Displays a summary of the virtual services, the status of the virtual services, and installed software packages.</td></tr>
</table>

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td><b>show guestshell detail</b>
<pre>
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></td><td>Displays details about the guestshell package (such as version, signing resources, and devices).</td></tr>
</table>

Persistently Starting Your Application From the Guest Shell

Your application should have a `systemd / systemctl` service file that gets installed in `/usr/lib/systemd/system/application_name.service`. This service file should have the following general format:

```
[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
```

Note To run `systemd` as a specific user, add `User=<username>` to the `[Service]` section of your service.

Procedure for Persistently Starting Your Application from the Guest Shell

Procedure

Step 1 Install your application service file that you created above into /usr/lib/systemd/system/*application_name*.service

Step 2 Start your application with **systemctl start** *application_name*

Step 3 Verify that your application is running with **systemctl status** -l *application_name*

Step 4 Enable your application to be restarted on reload with **systemctl enable** *application_name*

Step 5 Verify that your application is running with **systemctl status** -l *application_name*

An Example Application in the Guest Shell

The following example demonstrates an application in the Guest Shell:

```
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
```

After reload

```
[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]#
```

Running under `systemd / systemctl`, your application is automatically restarted if it dies (or if you kill it). The Process ID is originally 226. After killing the application, it is automatically restarted with a Process ID of 257.

```
[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]#
```

Troubleshooting Guest Shell Issues

Unable to Get Into Guest Shell After Downgrade to 7.0(3)I7

If you downgrade from the NX-OS 9.2(1) release to the NX-OS 7.0(3)7 release image (which does not have user namespace support) while the Guest Shell is in the process of activating or deactivating, you may run into the following condition where the Guest Shell activates, but you are unable to get into the Guest Shell. The reason for this issue is that if a reload is issued while the Guest Shell is in transition, the files within the Guest Shell can't get shifted back into an id range that is usable for NX-OS releases that don't have user namespace support.

```
switch# guestshell
Failed to mkdir .ssh for admin
admin RSA add failed
```

```
ERROR: Failed to connect with Virtual-service 'guestshell+'
switch#
switch# sh virt list

Virtual Service List:
Name           Status       Package Name
-----------------------------------------------------------------------
guestshell+    Activated    guestshell.ova

switch# run bash ls -al /isan/vdc_1/virtual-instance/guestshell+/rootfs/
drwxr-xr-x  24  11000  11000  1024  Apr 11  10:44  .
drwxrwxrwx   4  root   root     80  Apr 27  20:08  ..
-rw-r--r--   1  11000  11000     0  Mar 21  16:24  .autorelabel
lrwxrwxrwx   1  11000  11000     7  Mar 21  16:24  bin -> usr/bin
```

To recover from this issue without losing the contents of the Guest Shell, reload the system with the previously-running NX-OS 9.2(x) image and let the Guest Shell get to the `Activated` state before reloading the system with the NX-OS 7.0(3)I7 image. Another option is to disable the Guest Shell while running NX-OS 9.2(x) and re-enable it after reloading with 7.0(3)I7.

If you do not have anything to preserve in the Guest Shell and you just want to recover it, you can destroy and recreate it without needing to change images.

Unable to Access Files on bootflash from root in the Guest Shell

You may find that you are unable to access files on bootflash from root in the Guest Shell.

From the host:

```
root@switch# ls -al /bootflash/try.that
-rw-r--r--  1  root  root  0  Apr 27  20:55  /bootflash/try.that
root@switch#
```

From the Guest Shell:

```
[root@guestshellbootflash]# ls -al /bootflash/try.that
-rw-r--r--  1  65534  host-root  0  Apr 27  20:55  /bootflash/try.that
[root@guestshellbootflash]# echo "some text" >> /bootflash/try.that
-bash: /bootflash/try.that: Permission denied
[root@guestshellbootflash]#
```

This may be due to the fact that, because the user namespace is being used to protect the host system, root in the Guest Shell is not actually the root of the system.

To recover from this issue, verify that the file permissions and group-id of the files allow for shared files on bootflash to be accessed as expected. You may need to change the permissions or group-id from the host Bash session.