



## **Cisco Nexus 3600 NX-OS Programmability Guide, Release 9.2(x)**

**First Published:** 2018-07-18

**Last Modified:** 2019-07-24

### **Americas Headquarters**

Cisco Systems, Inc.  
170 West Tasman Drive  
San Jose, CA 95134-1706  
USA  
<http://www.cisco.com>  
Tel: 408 526-4000  
800 553-NETS (6387)  
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at [www.cisco.com/go/offices](http://www.cisco.com/go/offices).

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <https://www.cisco.com/go/trademarks>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1721R)

© 2018–2019 Cisco Systems, Inc. All rights reserved.



## CONTENTS

---

### PREFACE

<b>Preface</b>	<b>xiii</b>
Audience	xiii
Document Conventions	xiii
Related Documentation for Cisco Nexus 3600 Platform Switches	xiv
Documentation Feedback	xiv
Communications, Services, and Additional Information	xiv

---

### CHAPTER 1

<b>New and Changed Information</b>	<b>1</b>
New and Changed Information	1

---

### CHAPTER 2

<b>Overview</b>	<b>3</b>
Programmability Overview	3
Standard Network Manageability Features	4
Advanced Automation Feature	4
Power On Auto Provisioning Support	4
Programmability Support	4
NX-API Support	5
Python Scripting	5
Bash	5
Perl Modules	5

---

### PART I

<b>Shells and Scripting</b>	<b>7</b>
-----------------------------	----------

---

### CHAPTER 3

<b>Bash</b>	<b>9</b>
About Bash	9
Guidelines and Limitations	9

Accessing Bash	9
Escalate Privileges to Root	11
Examples of Bash Commands	12
Displaying System Statistics	12
Running Bash from CLI	13
Running Python from Bash	13
Managing RPMs	14
Installing RPMs from Bash	14
Upgrading RPMs	15
Downgrading an RPM	15
Erasing an RPM	15
Persistently Daemonizing an SDK- or ISO-built Third Party Process	16
Persistently Starting Your Application from the Native Bash Shell	17
An Example Application in the Native Bash Shell	17

---

**CHAPTER 4****Guest Shell 21**

About the Guest Shell	21
Guidelines and Limitations	22
Accessing the Guest Shell	26
Resources Used for the Guest Shell	27
Capabilities in the Guest Shell	27
NX-OS CLI in the Guest Shell	27
Network Access in Guest Shell	28
Access to Bootflash in Guest Shell	30
Python in Guest Shell	30
Python 3 in Guest Shell 2.x (Centos 7)	31
Installing RPMs in the Guest Shell	34
Security Posture for Guest Shell	35
Kernel Vulnerability Patches	35
ASLR and X-Space Support	35
Namespace Isolation	36
Root-User Restrictions	37
Resource Management	37
Guest File System Access Restrictions	38

Managing the Guest Shell	38
Disabling the Guest Shell	41
Destroying the Guest Shell	42
Enabling the Guest Shell	42
Replicating the Guest Shell	43
Exporting Guest Shell rootfs	44
Importing Guest Shell rootfs	44
Importing YAML File	45
show guestshell Command	49
Verifying Virtual Service and Guest Shell Information	49
Persistently Starting Your Application From the Guest Shell	51
Procedure for Persistently Starting Your Application from the Guest Shell	52
An Example Application in the Guest Shell	52
Troubleshooting Guest Shell Issues	53

---

**CHAPTER 5**
**Python API 55**

About the Python API	55
Using Python	55
Cisco Python Package	55
Using the CLI Command APIs	56
Invoking the Python Interpreter from the CLI	58
Display Formats	58
Non-interactive Python	59
Running Scripts with Embedded Event Manager	61
Python Integration with Cisco NX-OS Network Interfaces	61
Cisco NX-OS Security with Python	62
Examples of Security and User Authority	62
Example of Running Script with Scheduler	63

---

**CHAPTER 6**
**Scripting with Tcl 65**

About Tcl	65
Guidelines and Limitations	65
Tclsh Command Help	65
Tclsh Command History	66

- Tclsh Tab Completion 66
- Tclsh CLI Command 66
- Tclsh Command Separation 67
- Tcl Variables 67
- Tclquit 67
- Tclsh Security 67
- Running the Tclsh Command 68
- Navigating Cisco NX-OS Modes from the Tclsh Command 69
- Tcl References 70

---

**CHAPTER 7**      **iPXE 71**

- About iPXE 71
- Netboot Requirements 72
- Guidelines and Limitations 72
  - Notes for iPXE 72
- Boot Mode Configuration 80
- Verifying the Boot Order Configuration 82

---

**CHAPTER 8**      **Kernel Stack 83**

- About Kernel Stack 83
- Guidelines and Limitations 83
- Changing the Port Range 84

---

**PART II**          **Applications 87**

---

**CHAPTER 9**      **Third-Party Applications 89**

- About Third-Party Applications 89
- Installing Signed Third-Party RPMs by Importing Keys Automatically 89
- Installing Signed RPM 91
  - Checking a Signed RPM 91
  - Installing Signed RPMs by Manually Importing Key 92
  - Installing Signed Third-Party RPMs by Importing Keys Automatically 94
  - Adding Signed RPM into Repo 96
- Persistent Third-Party RPMs 96

Installing RPM from VSH	97
Package Addition	97
Package Activation	98
Deactivating Packages	99
Removing Packages	99
Displaying Installed Packages	99
Displaying Detail Logs	100
Upgrading a Package	100
Downgrading a Package	100
Third-Party Applications	101
NX-OS	101
collectd	101
Ganglia	101
Iperf	101
LLDP	102
Nagios	102
OpenSSH	102
Quagga	102
Splunk	102
tcollector	102
tcpdump	103
Tshark	103
<hr/>	
<b>CHAPTER 10</b>	<b>Ansible 105</b>
	Prerequisites 105
	About Ansible 105
	Cisco Ansible Module 105
<hr/>	
<b>CHAPTER 11</b>	<b>Puppet Agent 107</b>
	About Puppet 107
	Prerequisites 107
	Puppet Agent NX-OS Environment 108
	ciscopuppet Module 108

---

<b>CHAPTER 12</b>	<b>Using Chef Client with Cisco NX-OS</b>	<b>111</b>
	About Chef	111
	Prerequisites	111
	Chef Client NX-OS Environment	112
	cisco-cookbook	112

---

<b>CHAPTER 13</b>	<b>Nexus Application Development - ISO</b>	<b>115</b>
	About ISO	115
	Installing the ISO	115
	Using the ISO to Build Applications	116
	Using RPM to Package an Application	117

---

<b>CHAPTER 14</b>	<b>Nexus Application Development - SDK</b>	<b>119</b>
	About the Cisco SDK	119
	Installing the SDK	119
	Procedure for Installation and Environment Initialization	120
	Using the SDK to Build Applications	121
	Using RPM to Package an Application	122
	Creating an RPM Build Environment	123
	Using General RPM Build Procedure	123
	Example to Build RPM for collectd with No Optional Plug-Ins	124
	Example to Build RPM for collectd with Optional Curl Plug-In	125

---

<b>CHAPTER 15</b>	<b>NX-SDK</b>	<b>127</b>
	About the NX-SDK	127
	Install the NX-SDK	128
	Building and Packaging C++ Applications	128
	Installing and Running Custom Applications	131

---

<b>CHAPTER 16</b>	<b>Using Docker with Cisco NX-OS</b>	<b>135</b>
	About Docker with Cisco NX-OS	135
	Guidelines and Limitations	135



Prerequisites for Setting Up Docker Containers Within Cisco NX-OS	136
Starting the Docker Daemon	136
Configure Docker to Start Automatically	137
Starting Docker Containers: Host Networking Model	138
Starting Docker Containers: Bridged Networking Model	139
Mounting the bootflash and volatile Partitions in the Docker Container	140
Enabling Docker Daemon Persistence on Enhanced ISSU Switchover	140
Resizing the Docker Storage Backend	141
Stopping the Docker Daemon	143
Docker Container Security	144
Securing Docker Containers With User namespace Isolation	144
Moving the cgroup Partition	145
Docker Troubleshooting	145
Docker Fails to Start	146
Docker Fails to Start Due to Insufficient Storage	146
Failure to Pull Images from Docker Hub (509 Certificate Expiration Error Message)	147
Failure to Pull Images from Docker Hub (Client Timeout Error Message)	147
Docker Daemon or Containers Not Running On Switch Reload or Switchover	148
Resizing of Docker Storage Backend Fails	148
Docker Container Doesn't Receive Incoming Traffic On a Port	148
Unable to See Data Port And/Or Management Interfaces in Docker Container	149
General Troubleshooting Tips	149

---

**PART III**
**NX-API 151**


---

**CHAPTER 17**
**NX-API CLI 153**

About NX-API CLI	153
Transport	153
Message Format	154
Security	154
Using NX-API CLI	154
Escalate Privileges to Root on NX-API	156
NX-API Management Commands	157
Working With Interactive Commands Using NX-API	159

	NX-API Request Elements	159
	NX-API Response Elements	163
	Restricting Access to NX-API	164
	Updating an iptable	164
	Making an Iptable Persistent Across Reloads	166
	Table of NX-API Response Codes	167
	XML and JSON Supported Commands	168
	About JSON (JavaScript Object Notation)	169
	Examples of XML and JSON Output	169
<hr/>		
<b>CHAPTER 18</b>	<b>NX-API REST</b>	<b>177</b>
	About NX-API REST	177
<hr/>		
<b>CHAPTER 19</b>	<b>NX-API Developer Sandbox</b>	<b>179</b>
	NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2)	179
	About the NX-API Developer Sandbox	179
	Guidelines and Limitations	180
	Configuring the Message Format and Command Type	180
	Using the Developer Sandbox	182
	Using the Developer Sandbox to Convert CLI Commands to Payloads	182
	NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later	185
	About the NX-API Developer Sandbox	185
	Guidelines and Limitations	186
	Configuring the Message Format and Input Type	187
	Using the Developer Sandbox	189
	Using the Developer Sandbox to Convert CLI Commands to REST Payloads	189
	Using the Developer Sandbox to Convert from REST Payloads to CLI Commands	192
	Using the Developer Sandbox to Convert from RESTCONF to json or XML	197
<hr/>		
<b>PART IV</b>	<b>Model-Driven Programmability</b>	<b>201</b>
<hr/>		
<b>CHAPTER 20</b>	<b>Managing Components</b>	<b>203</b>
	About the Component RPM Packages	203
	Preparing For Installation	205

Downloading Components from the Cisco Artifactory	206
Installing RPM Packages	207
Installing the Programmable Interface Base And Common Model Component RPM Packages	207

---

**CHAPTER 21      Converting CLI Commands to Network Configuration Format    209**

Information About XMLIN	209
Licensing Requirements for XMLIN	209
Installing and Using the XMLIN Tool	210
Converting Show Command Output to XML	210
Configuration Examples for XMLIN	211

---

**PART V            XML Management Interface    215**

---

**CHAPTER 22      XML Management Interface    217**

About the XML Management Interface	217
About the XML Management Interface	217
NETCONF Layers	217
SSH xmlagent	218
Licensing Requirements for the XML Management Interface	218
Prerequisites to Using the XML Management Interface	219
Using the XML Management Interface	219
Configuring SSH and the XML Server Options	219
Starting an SSH Session	219
Sending the Hello Message	220
Obtaining the XSD Files	220
Sending an XML Document to the XML Server	221
Creating NETCONF XML Instances	221
RPC Request Tag rpc	222
NETCONF Operations Tags	223
Device Tags	224
Extended NETCONF Operations	226
NETCONF Replies	229
RPC Response Tag	230
Interpreting Tags Encapsulated in the Data Tag	230

Information About Example XML Instances	231
Example XML Instances	231
NETCONF Close Session Instance	231
NETCONF Kill-session Instance	232
NETCONF copy-config Instance	232
NETCONF edit-config Instance	232
NETCONF get-config Instance	234
NETCONF Lock Instance	234
NETCONF unlock Instance	235
NETCONF Commit Instance - Candidate Configuration Capability	236
NETCONF Confirmed-commit Instance	236
NETCONF rollback-on-error Instance	236
NETCONF validate Capability Instance	237
Additional References	237



## Preface

---

This preface includes the following sections:

- [Audience, on page xiii](#)
- [Document Conventions, on page xiii](#)
- [Related Documentation for Cisco Nexus 3600 Platform Switches, on page xiv](#)
- [Documentation Feedback, on page xiv](#)
- [Communications, Services, and Additional Information, on page xiv](#)

## Audience

This publication is for network administrators who install, configure, and maintain Cisco Nexus switches.

## Document Conventions

Command descriptions use the following conventions:

Convention	Description
<b>bold</b>	Bold text indicates the commands and keywords that you enter literally as shown.
<i>Italic</i>	Italic text indicates arguments for which the user supplies the values.
[x]	Square brackets enclose an optional element (keyword or argument).
[x   y]	Square brackets enclosing keywords or arguments separated by a vertical bar indicate an optional choice.
{x   y}	Braces enclosing keywords or arguments separated by a vertical bar indicate a required choice.
[x {y   z}]	Nested set of square brackets or braces indicate optional or required choices within optional or required elements. Braces and a vertical bar within square brackets indicate a required choice within an optional element.

Convention	Description
<i>variable</i>	Indicates a variable for which you supply values, in context where italics cannot be used.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Examples use the following conventions:

Convention	Description
<code>screen font</code>	Terminal sessions and information the switch displays are in screen font.
<b>boldface screen font</b>	Information you must enter is in boldface screen font.
<i>italic screen font</i>	Arguments for which you supply values are in italic screen font.
<>	Nonprinting characters, such as passwords, are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

## Related Documentation for Cisco Nexus 3600 Platform Switches

The entire Cisco Nexus 3600 platform switch documentation set is available at the following URL:

<http://www.cisco.com/c/en/us/support/switches/nexus-3000-series-switches/tsd-products-support-series-home.html>

## Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to [nexus3k-docfeedback@cisco.com](mailto:nexus3k-docfeedback@cisco.com). We appreciate your feedback.

## Communications, Services, and Additional Information

- To receive timely, relevant information from Cisco, sign up at [Cisco Profile Manager](#).
- To get the business impact you're looking for with the technologies that matter, visit [Cisco Services](#).
- To submit a service request, visit [Cisco Support](#).
- To discover and browse secure, validated enterprise-class apps, products, solutions and services, visit [Cisco Marketplace](#).
- To obtain general networking, training, and certification titles, visit [Cisco Press](#).
- To find warranty information for a specific product or product family, access [Cisco Warranty Finder](#).

### **Cisco Bug Search Tool**

[Cisco Bug Search Tool](#) (BST) is a web-based tool that acts as a gateway to the Cisco bug tracking system that maintains a comprehensive list of defects and vulnerabilities in Cisco products and software. BST provides you with detailed defect information about your products and software.







# CHAPTER 1

## New and Changed Information

This chapter provides release-specific information for each new and changed feature in the *Cisco Nexus 3600 Series NX-OS Programmability Guide, 9.2(x)*.

- [New and Changed Information, on page 1](#)

## New and Changed Information

This table summarizes the new and changed features for the *Cisco Nexus 3600 Series NX-OS Programmability Guide, Release 9.2(x)* and where they are documented.

**Table 1: New and Changed Features**

Feature	Description	Changed in Release	Where Documented
	Changed the document title from 9.x to 9.2(x)		Title page
Support for XML and JSON output	Support for displaying the output of Cisco NX-OS <b>show</b> commands is now extended to the Cisco Nexus 3600-R switches.	9.2(3)	<a href="#">NX-API CLI, on page 153</a>
Support for CLI commands for NETCONF	Converting NX-OS CLI commands to Network Configuration format is documented.	9.2(2)	<a href="#">Converting CLI Commands to Network Configuration Format, on page 209</a>
XML Management Interface	Support for managing the Cisco Nexus 3600 switches with an XML-based tool through the XML-based Network Configuration Protocol (NETCONF) is documented.	9.2(2)	<a href="#">XML Management Interface, on page 217</a>

<b>Feature</b>	<b>Description</b>	<b>Changed in Release</b>	<b>Where Documented</b>
Updates to NX-API Sandbox	Various enhancements have been added to the NX-API Developer Sandbox.	9.2(2)	<a href="#">NX-API Developer Sandbox, on page 179</a>
Perl modules	Support for the Perl modules has been added.	9.2(2)	<a href="#">Overview, on page 3</a>
No updates since Cisco NX-OS Release 7.x.	First 9.x release	9.2(1)	



## CHAPTER 2

# Overview

---

- [Programmability Overview, on page 3](#)
- [Standard Network Manageability Features, on page 4](#)
- [Advanced Automation Feature, on page 4](#)
- [Programmability Support, on page 4](#)

## Programmability Overview

The Cisco NX-OS software running on the Cisco Nexus 3600 Series devices is as follows:

- **Resilient**  
Provides critical business-class availability.
- **Modular**  
Has extensions that accommodate business needs.
- **Highly Programmatic**  
Allows for rapid automation and orchestration through Application Programming Interfaces (APIs).
- **Secure**  
Protects and preserves data and operations.
- **Flexible**  
Integrates and enables new technologies.
- **Scalable**  
Accommodates and grows with the business and its requirements.
- **Easy to use**  
Reduces the amount of learning required, simplifies deployment, and provides ease of manageability.

With the Cisco NX-OS operating system, the device functions in the unified fabric mode to provide network connectivity with programmatic automation functions.

Cisco NX-OS contains Open Source Software (OSS) and commercial technologies that provide automation, orchestration, programmability, monitoring and compliance support.

For more information on Open NX-OS, see <https://developer.cisco.com/site/nx-os/>.

## Standard Network Manageability Features

- SNMP (V1, V2, V3)
- Syslog
- RMON
- NETCONF
- CLI and CLI scripting

## Advanced Automation Feature

The enhanced Cisco NX-OS on the device supports automation. The platform includes support for Power On Auto Provisioning (POAP).

The enhanced Cisco NX-OS on the device supports automation. The platform includes the following features that support automation:

- Power On Auto Provisioning (POAP) support
- Chef and Puppet integration
- OpenStack integration
- OpenDayLight integration and OpenFlow support

## Power On Auto Provisioning Support

Power On Auto Provisioning (POAP) automates the process of installing and upgrading software images and installing configuration files on Cisco Nexus devices that are being deployed in the network for the first time. It reduces the manual tasks that are required to scale the network capacity.

When a Cisco Nexus device with the POAP feature boots and does not find the startup configuration, the device enters POAP mode. It locates a DHCP server and bootstraps itself with its interface IP address, gateway, and DNS server IP addresses. The device obtains the IP address of a TFTP server or the URL of an HTTP server and downloads a configuration script that enables the device to download and install the appropriate software image and configuration file.

## Programmability Support

Cisco NX-OS on Cisco Nexus 9000 devices support several capabilities to aid programmability.

## NX-API Support

Cisco NX-API allows for HTTP-based programmatic access to the Cisco Nexus 9000 platform. This support is delivered by NX-API, an open source webservice. NX-API provides the configuration and management capabilities of the Cisco NX-OS CLI with web-based APIs. The device can be set to publish the output of the API calls in XML or JSON format. This API enables rapid development on the Cisco Nexus 9000 platform.

## Python Scripting

Cisco Nexus 9000 devices support Python v2.7.5 in both interactive and noninteractive (script) modes.

The Python scripting capability on the devices provides programmatic access to the switch CLI to perform various tasks, and to Power-On Auto Provisioning (POAP) and Embedded Event Manager (EEM) actions. Responses to Python calls that invoke the Cisco NX-OS CLI return text or JSON output.

The Python interpreter is included in the Cisco NX-OS software.

## Bash

Cisco Nexus 9000 devices support direct Bourne-Again Shell (Bash) access. With Bash, you can access the underlying Linux system on the device and manage the system.

## Perl Modules

In order to support additional applications, the following Perl modules have been added:

- bytes.pm
- feature.pm
- hostname.pl
- lib.pm
- overload.pm
- Carp.pm
- Class/Struct.pm
- Data/Dumper.pm
- DynaLoader.pm
- Exporter/Heavy.pm
- FileHandle.pm
- File/Basename.pm
- File/Glob.pm
- File/Spec.pm
- File/Spec/Unix.pm

- File/stat.pm
- Getopt/Std.pm
- IO.pm
- IO/File.pm
- IO/Handle.pm
- IO/Seekable.pm
- IO/Select.pm
- List/Util.pm
- MIME/Base64.pm
- SelectSaver.pm
- Socket.pm
- Symbol.pm
- Sys/Hostname.pm
- Time/HiRes.pm
- auto/Data/Dumper/Dumper.so
- auto/File/Glob/Glob.so
- auto/IO/IO.so
- auto/List/Util/Util.so
- auto/MIME/Base64/Base64.so
- auto/Socket/Socket.so
- auto/Sys/Hostname/Hostname.so
- auto/Time/HiRes/HiRes.so



## PART I

# Shells and Scripting

- [Bash, on page 9](#)
- [Guest Shell, on page 21](#)
- [Python API, on page 55](#)
- [Scripting with Tcl, on page 65](#)
- [iPXE, on page 71](#)
- [Kernel Stack, on page 83](#)







## CHAPTER 3

# Bash

---

- [About Bash, on page 9](#)
- [Guidelines and Limitations, on page 9](#)
- [Accessing Bash, on page 9](#)
- [Escalate Privileges to Root, on page 11](#)
- [Examples of Bash Commands, on page 12](#)
- [Managing RPMs, on page 14](#)
- [Persistently Daemonizing an SDK- or ISO-built Third Party Process, on page 16](#)
- [Persistently Starting Your Application from the Native Bash Shell, on page 17](#)
- [An Example Application in the Native Bash Shell, on page 17](#)

## About Bash

In addition to the NX-OS CLI, Cisco Nexus 3600 devices support access to the Bourne-Again SHell (Bash). Bash interprets commands that you enter or commands that are read from a shell script. Using Bash enables access to the underlying Linux system on the device and to manage the system.

## Guidelines and Limitations

The Bash shell has the following guidelines and limitations:

- The binaries located in the `/isan` folder are meant to be run in an environment which is setup differently from that of the shell entered from the **run bash** command. It is advisable not to use these binaries from the Bash shell as the behavior within this environment is not predictable.

## Accessing Bash

In Cisco NX-OS, Bash is accessible from user accounts that are associated with the Cisco NX-OS dev-ops role or the Cisco NX-OS network-admin role.

The following example shows the authority of the dev-ops role and the network-admin role:

```
switch# show role name dev-ops
```

```
Role: dev-ops
```

```

Description: Predefined system role for devops access. This role
cannot be modified.
Vlan policy: permit (default)
Interface policy: permit (default)
Vrf policy: permit (default)
-----

```

Rule	Perm	Type	Scope	Entity
4	permit	command		conf t ; username *
3	permit	command		bcm module *
2	permit	command		run bash *
1	permit	command		python *

```
switch# show role name network-admin
```

```
Role: network-admin
```

```
Description: Predefined network admin role has access to all commands
on the switch
-----
```

Rule	Perm	Type	Scope	Entity
1	permit	read-write		

```
switch#
```

Bash is enabled by running the **feature bash-shell** command.

The **run bash** command loads Bash and begins at the home directory for the user.

The following examples show how to enable the Bash shell feature and how to run Bash.

```
switch# configure terminal
switch(config)# feature bash-shell
```

```
switch# run?
run          Execute/run program
run-script   Run shell scripts
```

```
switch# run bash?
bash        Linux-bash
```

```
switch# run bash
bash-4.2$ whoami
admin
bash-4.2$ pwd
/bootflash/home/admin
bash-4.2$
```




---

**Note** You can also execute Bash commands with **run bash** *command*.

For instance, you can run **whoami** using **run bash** *command*:

```
run bash whoami
```

You can also run Bash by configuring the user **shelltype**:

```
username foo shelltype bash
```

This command puts you directly into the Bash shell.

---

# Escalate Privileges to Root

The privileges of an admin user can escalate their privileges for root access.

The following are guidelines for escalating privileges:

- Only an admin user can escalate privileges to root.
- Bash must be enabled before escalating privileges.
- Escalation to root is password protected.
- SSH to the switch using `root` username through a non-management interface will default to Linux Bash shell-type access for the root user. Type `vsh` to return to NX-OS shell access.

NX-OS network administrator users must escalate to root to pass configuration commands to the NX-OS VSH if:

- The NX-OS user has a shell-type Bash and logs into the switch with a shell-type Bash.
- The NX-OS user logged into the switch in Bash continues to use Bash on the switch.

Run `sudo su 'vsh -c "<configuration commands>"` or `sudo bash -c 'vsh -c "<configuration commands>"`.

The example below demonstrates with network administrator user MyUser with a default shelltype Bash using `sudo` to pass configuration commands to the NX-OS:

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ sudo vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show
interface eth1/2 brief"
```

```
-----
Ethernet      VLAN      Type Mode      Status Reason                               Speed      Port
Interface                                           Ch #
-----
Eth1/2        --        eth  routed down  Administratively down                auto(D)  --
```

The example below demonstrates with network administrator user MyUser with default shelltype Bash entering the NX-OS and then running Bash on the NX-OS:

```
ssh -l MyUser 1.2.3.4
-bash-4.2$ vsh -h
Cisco NX-OS Software
Copyright (c) 2002-2016, Cisco Systems, Inc. All rights reserved.
Nexus 3600 software ("Nexus 3600 Software") and related documentation,
files or other reference materials ("Documentation") are
the proprietary property and confidential information of Cisco
Systems, Inc. ("Cisco") and are protected, without limitation,
pursuant to United States and International copyright and trademark
laws in the applicable jurisdiction which provide civil and criminal
penalties for copying or distribution without Cisco's authorization.
```

Any use or disclosure, in whole or in part, of the Nexus 3600 Software or Documentation to any third party for any purposes is expressly prohibited except as otherwise authorized by Cisco in writing. The copyrights to certain works contained herein are owned by other third parties and are used and distributed under license. Some parts of this software may be covered under the GNU Public License or the GNU Lesser General Public License. A copy of each such license is available at

```

http://www.gnu.org/licenses/gpl.html and
http://www.gnu.org/licenses/lgpl.html
*****
* Nexus 3600 is strictly limited to use for evaluation, demonstration *
* and NX-OS education. Any use or disclosure, in whole or in part of *
* the Nexus 3600 Software or Documentation to any third party for any *
* purposes is expressly prohibited except as otherwise authorized by *
* Cisco in writing. *
*****
switch# run bash
bash-4.2$ vsh -c "configure terminal ; interface eth1/2 ; shutdown ; sleep 2 ; show interface
eth1/2 brief"

```

```

-----
Ethernet      VLAN      Type Mode      Status Reason                               Speed      Port
Interface
-----
Eth1/2        --        eth  routed down  Administratively down                auto(D) --

```

The following example shows how to escalate privileges to root and how to verify the escalation:

```

switch# run bash
bash-4.2$ sudo su root
bash-4.2# whoami
root
bash-4.2# exit
exit

```

## Examples of Bash Commands

This section contains examples of Bash commands and output.

## Displaying System Statistics

The following example displays system statistics:

```

switch# run bash
bash-4.2$ cat /proc/meminfo
<snip>
MemTotal:      16402560 kB
MemFree:       14098136 kB
Buffers:       11492 kB
Cached:        1287880 kB
SwapCached:    0 kB
Active:        1109448 kB
Inactive:      717036 kB
Active(anon):  817856 kB
Inactive(anon): 702880 kB
Active(file):  291592 kB
Inactive(file): 14156 kB
Unevictable:   0 kB
Mlocked:      0 kB
SwapTotal:     0 kB
SwapFree:      0 kB
Dirty:         32 kB
Writeback:     0 kB
AnonPages:    527088 kB

```

```
Mapped:          97832 kB
<\snip>
```

## Running Bash from CLI

The following example runs **ps** from Bash using **run bash** command:

```
switch# run bash ps -el
F S  UID  PID  PPID  C  PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  0    1    0  0  80   0 -  528 poll_s ?          00:00:03 init
1 S  0    2    0  0  80   0 -   0 kthrea ?          00:00:00 kthreadd
1 S  0    3    2  0  80   0 -   0 run_ks ?          00:00:56 ksoftirqd/0
1 S  0    6    2  0 -40  - -   0 cpu_st ?          00:00:00 migration/0
1 S  0    7    2  0 -40  - -   0 watchd ?          00:00:00 watchdog/0
1 S  0    8    2  0 -40  - -   0 cpu_st ?          00:00:00 migration/1
1 S  0    9    2  0  80   0 -   0 worker ?          00:00:00 kworker/1:0
1 S  0   10    2  0  80   0 -   0 run_ks ?          00:00:00 ksoftirqd/1
```

## Running Python from Bash

The following example shows how to load Python and configure a switch using Python objects:

```
switch# run bash
bash-4.2$ python
Python 2.7.5 (default, Oct 8 2013, 23:59:43)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cisco import *
>>> from cisco.vrf import *
>>> from cisco.interface import *
>>> vrfobj=VRF('myvrf')
>>> vrfobj.get_name()
'myvrf'
>>> vrfobj.add_interface('Ethernet1/3')
True
>>> intf=Interface('Ethernet1/3')
>>> print intf.config()

!Command: show running-config interface Ethernet1/3
!Time: Mon Nov 4 13:17:56 2013

version 6.1(2)I2(1)

interface Ethernet1/3
  vrf member myvrf

>>>
```

# Managing RPMs

## Installing RPMs from Bash

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<code>sudo yum installed   grep platform</code>	Displays a list of the NX-OS feature RPMs installed on the switch.
<b>Step 2</b>	<code>sudo yum list available</code>	Displays a list of the available RPMs.
<b>Step 3</b>	<code>sudo yum -y install rpm</code>	Installs an available RPM.

### Example

The following is an example of installing the **bfd** RPM:

```
bash-4.2$ sudo yum list installed | grep n3600
base-files.n3600                3.0.14-r74.2                installed
bfd.lib32_n3600                1.0.0-r0                    installed
core.lib32_n3600              1.0.0-r0                    installed
eigrp.lib32_n3600             1.0.0-r0                    installed
eth.lib32_n3600               1.0.0-r0                    installed
isis.lib32_n3600              1.0.0-r0                    installed
lACP.lib32_n3600              1.0.0-r0                    installed
linecard.lib32_n3600          1.0.0-r0                    installed
lldp.lib32_n3600              1.0.0-r0                    installed
ntp.lib32_n3600               1.0.0-r0                    installed
nxos-ssh.lib32_n3600          1.0.0-r0                    installed
ospf.lib32_n3600              1.0.0-r0                    installed
perf-cisco.n3600_gdb           3.12-r0                     installed
platform.lib32_n3600          1.0.0-r0                    installed
shadow-securetty.n3600_gdb    4.1.4.3-r1                  installed
snmp.lib32_n3600              1.0.0-r0                    installed
svi.lib32_n3600               1.0.0-r0                    installed
sysvinit-inittab.n3600_gdb    2.88dsf-r14                 installed
tacacs.lib32_n3600            1.0.0-r0                    installed
task-nxos-base.n3600_gdb      1.0-r0                       installed
tor.lib32_n3600               1.0.0-r0                    installed
vtp.lib32_n3600               1.0.0-r0                    installed
bash-4.2$ sudo yum list available
bgp.lib32_n3600                1.0.0-r0
bash-4.2$ sudo yum -y install bfd
```



### Note

Upon switch reload during boot up, use the **rpm** command instead of **yum** for persistent RPMs. Otherwise, RPMs initially installed using **yum bash** or **install CLI** will show **reponame** or **filename** instead of **installed**.

## Upgrading RPMs

### Before you begin

There must be a higher version of the RPM in the Yum repository.

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<code>sudo yum -y upgrade rpm</code>	Upgrades an installed RPM.

### Example

The following is an example of upgrading the **bfd** RPM:

```
bash-4.2$ sudo yum -y upgrade bfd
```

## Downgrading an RPM

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<code>sudo yum -y downgrade rpm</code>	Downgrades the RPM if any of the Yum repositories has a lower version of the RPM.

### Example

The following example shows how to downgrade the **bfd** RPM:

```
bash-4.2$ sudo yum -y downgrade bfd
```

## Erasing an RPM



### Note

The SNMP RPM and the NTP RPM are protected and cannot be erased.

You can upgrade or downgrade these RPMs. It requires a system reload for the upgrade or downgrade to take effect.

For the list of protected rpms, see `/etc/yum/protected.d/protected_pkgs.conf`.

**Procedure**

	Command or Action	Purpose
<b>Step 1</b>	<code>sudo yum -y erase rpm</code>	Erases the RPM.

**Example**

The following example shows how to erase the **bfd** RPM:

```
bash-4.2$ sudo yum -y erase bfd
```

## Persistently Daemonizing an SDK- or ISO-built Third Party Process

Your application should have a startup bash script that gets installed in `/etc/init.d/application_name`. This startup bash script should have the following general format (for more information on this format, see <http://linux.die.net/man/8/chkconfig>).

```
#!/bin/bash
#
# <application_name> Short description of your application
#
# chkconfig: 2345 15 85
# description: Short description of your application
#
### BEGIN INIT INFO
# Provides: <application_name>
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Short description of your application
### END INIT INFO
# See how we were called.
case "$1" in
start)
# Put your startup commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
stop)
# Put your stop commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
status)
# Put your status commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
restart|force-reload|reload)
# Put your restart commands here
# Set RETVAL to 0 for success, non-0 for failure
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL
```



# Persistently Starting Your Application from the Native Bash Shell

## Procedure

- 
- Step 1** Install your application startup bash script that you created above into `/etc/init.d/application_name`
  - Step 2** Start your application with `/etc/init.d/application_name start`
  - Step 3** Enter `chkconfig --add application_name`
  - Step 4** Enter `chkconfig --level 3 application_name on`  
Run level 3 is the standard multi-user run level, and the level at which the switch normally runs.
  - Step 5** Verify that your application is scheduled to run on level 3 by running `chkconfig --list application_name` and confirm that level 3 is set to on
  - Step 6** Verify that your application is listed in `/etc/rc3.d`. You should see something like this, where there is an 'S' followed by a number, followed by your application name (tcollector in this example), and a link to your bash startup script in `../init.d/application_name`
- 

```
bash-4.2# ls -l /etc/rc3.d/tcollector
lrwxrwxrwx 1 root root 20 Sep 25 22:56 /etc/rc3.d/S15tcollector -> ../init.d/tcollector
bash-4.2#
```

## An Example Application in the Native Bash Shell

The following example demonstrates an application in the Native Bash Shell:

```
bash-4.2# cat /etc/init.d/hello.sh
#!/bin/bash

PIDFILE=/tmp/hello.pid
OUTPUTFILE=/tmp/hello

echo $$ > $PIDFILE
rm -f $OUTPUTFILE
while true
do
    echo $(date) >> $OUTPUTFILE
    echo 'Hello World' >> $OUTPUTFILE
    sleep 10
done
bash-4.2#
bash-4.2#
bash-4.2# cat /etc/init.d/hello
#!/bin/bash
#
# hello Trivial "hello world" example Third Party App
#
```

```

# chkconfig: 2345 15 85
# description: Trivial example Third Party App
#
### BEGIN INIT INFO
# Provides: hello
# Required-Start: $local_fs $remote_fs $network $named
# Required-Stop: $local_fs $remote_fs $network
# Description: Trivial example Third Party App
### END INIT INFO

PIDFILE=/tmp/hello.pid

# See how we were called.
case "$1" in
start)
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
stop)
    kill -9 `cat $PIDFILE`
    RETVAL=$?
;;
status)
    ps -p `cat $PIDFILE`
    RETVAL=$?
;;
restart|force-reload|reload)
    kill -9 `cat $PIDFILE`
    /etc/init.d/hello.sh &
    RETVAL=$?
;;
*)
echo $"Usage: $prog {start|stop|status|restart|force-reload}"
RETVAL=2
esac

exit $RETVAL
bash-4.2#
bash-4.2# chkconfig --add hello
bash-4.2# chkconfig --level 3 hello on
bash-4.2# chkconfig --list hello
hello          0:off  1:off  2:on   3:on   4:on   5:on   6:off
bash-4.2# ls -al /etc/rc3.d/*hello*
lrwxrwxrwx 1 root root 15 Sep 27 18:00 /etc/rc3.d/S15hello -> ../init.d/hello
bash-4.2#
bash-4.2# reboot

```

#### After reload

```

bash-4.2# ps -ef | grep hello
root      8790      1   0 18:03 ?           00:00:00 /bin/bash /etc/init.d/hello.sh
root      8973     8775   0 18:04 ttyS0       00:00:00 grep hello
bash-4.2#
bash-4.2# ls -al /tmp/hello*
-rw-rw-rw- 1 root root 205 Sep 27 18:04 /tmp/hello
-rw-rw-rw- 1 root root   5 Sep 27 18:03 /tmp/hello.pid
bash-4.2# cat /tmp/hello.pid
8790
bash-4.2# cat /tmp/hello
Sun Sep 27 18:03:49 UTC 2015
Hello World
Sun Sep 27 18:03:59 UTC 2015
Hello World
Sun Sep 27 18:04:09 UTC 2015
Hello World

```

```
Sun Sep 27 18:04:19 UTC 2015
Hello World
Sun Sep 27 18:04:29 UTC 2015
Hello World
Sun Sep 27 18:04:39 UTC 2015
Hello World
bash-4.2#
```





## CHAPTER 4

# Guest Shell

---

- [About the Guest Shell, on page 21](#)
- [Guidelines and Limitations, on page 22](#)
- [Accessing the Guest Shell, on page 26](#)
- [Resources Used for the Guest Shell, on page 27](#)
- [Capabilities in the Guest Shell, on page 27](#)
- [Security Posture for Guest Shell, on page 35](#)
- [Guest File System Access Restrictions , on page 38](#)
- [Managing the Guest Shell, on page 38](#)
- [Verifying Virtual Service and Guest Shell Information, on page 49](#)
- [Persistently Starting Your Application From the Guest Shell, on page 51](#)
- [Procedure for Persistently Starting Your Application from the Guest Shell, on page 52](#)
- [An Example Application in the Guest Shell, on page 52](#)
- [Troubleshooting Guest Shell Issues, on page 53](#)

## About the Guest Shell

In addition to the NX-OS CLI and Bash access on the underlying Linux environment, the Cisco Nexus 3000 Series devices 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 Cisco Nexus 3000 bootflash.
- Access to Cisco Nexus 3000 volatile tmpfs.
- Access to Cisco Nexus 3000 CLI.
- 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 35 MB of RAM and 200 MB of bootflash when enabled. Use the **guestshell destroy** command to reclaim resources if the Guest Shell is not used.

---

## Guidelines and Limitations

### Common Guidelines Across All Releases




---

**Important** If you have performed custom work inside your installation of the Guest Shell, save your changes to bootflash, off-box storage, or elsewhere outside the Guest Shell root file system before performing an upgrade.

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

- Use the `run guestshell` CLI command to access the Guest Shell on the Cisco Nexus device: The `run guestshell` command parallels the `run bash` command used to access the host shell. This command allows you to access the Guest Shell and get a bash prompt or run a command within the context of the Guest Shell. The command uses password-less SSH to an available port on the localhost in the default network namespace.
- `sshd` utility can secure the pre-configured SSH access into the Guest Shell by listening on `localhost` to avoid connection attempts from outside the network. `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 Guest Shell after Guest Shell restarts.
  - Only `root` can read the file that contains the key on the host to prevent a non-privileged user with host bash access from being able to use the key to connect to the Guest Shell. Network-admin users may start another instance of `sshd` in the Guest Shell to allow remote access directly into the Guest Shell, but any user that logs into the Guest Shell is also given network-admin privilege




---

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

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

Guest Shell installations prior to 2.2 (0.2) will not dynamically create individual user accounts.

---

- Installing the Cisco Nexus series switch software release on a fresh out-of-the-box Cisco Nexus switch will automatically enable the Guest Shell. Subsequent upgrades to the Cisco Nexus series switch software will NOT automatically upgrade Guest Shell.
- Guest Shell releases increment the major number when distributions or distribution versions change.
- Guest Shell releases increment the minor number when CVEs have been addressed. The Guest Shell will update CVEs only when CentOS makes them publically 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 Guest Shell rootfs. Any customizations and software package installations would then need to be performed again within the context of this new Guest Shell rootfs.

### Upgrading from Guest Shell 1.0 to Guest Shell 2.x

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

### Guest Shell 2.x

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



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

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: Guest shell 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 Guest Shell with the **guestshell destroy** command before reloading an older Cisco Nexus image that does not support the Guest Shell.

---

### Pre-Configured SSHD Service

The Guest Shell 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 Guest Shell from the NX-OS vegas-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 Guest Shell from the NX-OS CLI might not work.

The following steps instantiate an OpenSSH server within the Guest Shell as root:

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




---

**Note** The Guest Shell 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/system/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`.

```
[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
```

3. 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
```

4. 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
```

5. (optional) Check the configuration.

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

6. SSH into Guest Shell:

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

7. Save the configuration across multiple Guest Shell or switch reboots.

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

8. 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
```

9. 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/*
```

10. 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 Guest Shell 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 Guest Shell 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, the Guest Shell is accessible to the network-admin. 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** When running in the Guest Shell, you have network-admin level privileges.



**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* | *roofs*} command changes these limits.

Resource	Default	Minimum/Maximum
CPU	1%	1/6%
Memory	256MB	256/3840MB
Storage	200MB	200/2000MB

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 Guest Shell

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

The Guest Shell is populated with CentOS 7 Linux which provides the ability to Yum install software packages built for this distribution. The Guest Shell 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. Python 2.7.5 is included by default as is the PIP for installing additional python packages.

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

The Guest Shell 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 **vrinfo**, 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 7 environments, including the Guest Shell.

## 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** Starting 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 Cisco Nexus 3000 device 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 2.x (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-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 blfb 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/51/1d/23c926cc841ea67cd02a00aba99ae0f828e89d72b2190f27c11d4b7fb/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/cd551d81d8e15200e1cf41cd03869a46fe7226e7450af7a6545b0c474c9/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/cc/a9/01ffbf562e4274b6487b4bb1dbbc7ca55c7510b22e4c51f1409844368/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/d50cac98a05b006c55a399c3f1db9db7b5a24b7890bc9cf5db9e99/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/e8/74/6e4f91745020f967d0932b2b289b1009095734692ab88a4fe91b77f/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.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.e17
rh-python35-runtime.x86_64 0:2.0-2.e17

```

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, go to the CentOS 7 repo at [http://mirror.centos.org/centos/7/os/x86\\_64/Packages/](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 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 Guest Shell

Use of the Guest Shell in Cisco Nexus 3000 series devices 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 Nexus 3000 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 the 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 the Guest Shell follow:

- `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`

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 between Guest Shell and services on the host.

## Guest File System Access Restrictions

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 2: Guest Shell CLI Commands**

Commands	Description
<code>guestshell enable {package [guest shell OVA file   rootfs-file-URI]}</code>	<ul style="list-style-type: none"> <li>When <i>guest shell OVA file</i> 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 <b>guestshell disable</b> command.</li> <li>When <i>rootfs-file-URI</i> is specified: Imports a Guest Shell <b>rootfs</b> when the Guest Shell is in a destroyed state. This command brings up the Guest Shell with the specified package.</li> </ul>
<code>guestshell export rootfs package destination-file-URI</code>	Exports a Guest Shell <b>rootfs</b> file to a local URI (bootflash, USB1, etc.).
<code>guestshell disable</code>	Shuts down and disables the Guest Shell.

Commands	Description
<p><b>guestshell upgrade</b> {<b>package</b> [<i>guest shell OVA file</i>   <i>rootfs-file-URI</i>]}</p>	<ul style="list-style-type: none"> <li>When <i>guest shell OVA file</i> is specified: <p>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.</p> <p>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 <b>guestshell destroy</b> command followed by a <b>guestshell enable</b> command could also be used to replace the rootfs. When an upgrade is successful, the Guest Shell is activated.</p> <p>You are prompted for a confirmation prior to carrying out the upgrade command.</p> </li> <li>When <i>rootfs-file-URI</i> is specified: <p>Imports a Guest Shell <b>rootfs</b> file when the Guest Shell is already installed. This command removes the existing Guest Shell and installs the specified package.</p> </li> </ul>
<p><b>guestshell reboot</b></p>	<p>Deactivates the Guest Shell and then reactivates it. You are prompted for a confirmation prior to carrying out the reboot command.</p> <p><b>Note</b> This is the equivalent of a <b>guestshell disable</b> command followed by a <b>guestshell enable</b> command in exec mode.</p> <p>This is useful when processes inside the Guest Shell have been stopped and need to be restarted. The <b>run guestshell</b> command relies on <code>sshd</code> running in the Guest Shell.</p> <p>If the command does not work, the <code>sshd</code> 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.</p>

Commands	Description
<b>guestshell destroy</b>	<p>Deactivates and uninstalls the Guest Shell. All resources associated with the Guest Shell are returned to the system. The <b>show virtual-service global</b> command indicates when these resources become available.</p> <p>Issuing this command results in a prompt for a confirmation prior to carrying out the destroy command.</p>
<b>guestshell</b> <b>run guestshell</b>	Connects to the Guest Shell that is already running with a shell prompt. No username/password is required.
<b>guestshell run</b> <i>command</i> <b>run guestshell</b> <i>command</i>	<p>Executes a Linux/UNIX command within the context of the Guest Shell environment.</p> <p>After execution of the command you are returned to the switch prompt.</p>
<b>guestshell resize</b> [cpu   memory   rootfs]	<p>Changes the allotted resources available for the Guest Shell. The changes take effect the next time the Guest Shell is enabled or rebooted.</p> <p><b>Note</b>     Resize values are cleared when the <b>guestshell destroy</b> command is used.</p>
<b>guestshell sync</b>	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.
<b>virtual-service reset force</b>	<p>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.</p> <p>You are prompted for a confirmation prior to initiating the reset.</p>






---

**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, LXC's 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 [https://github.com/datacenter/opennxos/tree/master/guestshell\\_import\\_export](https://github.com/datacenter/opennxos/tree/master/guestshell_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:

Command	Description
<pre> <b>show virtual-service global</b>  switch# <b>show virtual-service global</b>  Virtual Service Global State and Virtualization Limits:  Infrastructure version : 1.9 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>Displays the global state and limits for virtual services.</p>
<pre> <b>show virtual-service list</b>  switch# <b>show virtual-service list *</b>  Virtual Service List:  Name                Status           Package Name ----- guestshell+         Activated        guestshell.ova </pre>	<p>Displays a summary of the virtual services, the status of the virtual services, and installed software packages.</p>

Command	Description
<pre> <b>show guestshell detail</b>  switch# <b>show guestshell detail</b> 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                : 250 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>Displays details about the guestshell package (such as version, signing resources, and devices).</p>

## 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.



## CHAPTER 5

# Python API

---

- [About the Python API](#) , on page 55
- [Using Python](#), on page 55

## About the Python API

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python website:

<http://www.python.org/>

The same site also contains distributions of and pointers to many free third-party Python modules, programs and tools, and additional documentation.

The Cisco Nexus 3600 devices support Python v2.7.5 in both interactive and non-interactive (script) modes and is available in the Guest Shell.

The Python scripting capability gives programmatic access to the device's command-line interface (CLI) to perform various tasks and PowerOn Auto Provisioning (POAP) or Embedded Event Manager (EEM) actions. Python can also be accessed from the Bash shell.

The Python interpreter is available in the Cisco NX-OS software.

## Using Python

This section describes how to write and execute Python scripts.

## Cisco Python Package

Cisco NX-OS provides a Cisco Python package that enables access to many core network device modules, such as interfaces, VLANs, VRFs, ACLs and routes. You can display the details of the Cisco Python package by entering the **help()** command. To obtain additional information about the classes and methods in a module,

you can run the help command for a specific module. For example, `help(cisco.interface)` displays the properties of the `cisco.interface` module.

The following is an example of how to display information about the Cisco python package:

```
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
    cisco

FILE
    /isan/python/scripts/cisco/__init__.py

PACKAGE CONTENTS
    acl
    bgp
    cisco_secret
    cisco_socket
    feature
    interface
    key
    line_parser
    md5sum
    nxcli
    ospf
    routemap
    routes
    section_parser
    ssh
    system
    tacacs
    vrf

CLASSES
    __builtin__.object
    cisco.cisco_secret.CiscoSecret
    cisco.interface.Interface
    cisco.key.Key
```

## Using the CLI Command APIs

The Python programming language uses three APIs that can execute CLI commands. The APIs are available from the Python CLI module.

These APIs are listed in the following table. You need to enable the APIs with the `from cli import *` command. The arguments for these APIs are strings of CLI commands. To execute a CLI command through the Python interpreter, you enter the CLI command as an argument string of one of the following APIs:



Table 3: CLI Command APIs

API	Description
<b>cli()</b> Example: <pre>string = cli ("cli-command")</pre>	Returns the raw output of CLI commands, including control/special characters.  <b>Note</b> The interactive Python interpreter prints control/special characters 'escaped'. A carriage return is printed as '\n' and gives results that might be difficult to read. The <b>clip()</b> API gives results that are more readable.
<b>clid()</b> Example: <pre>json_string = clid ("cli-command")</pre>	Returns JSON output for cli-command, if XML support exists for the command, otherwise an exception is thrown.  <b>Note</b> This API can be useful when searching the output of show commands.
<b>clip()</b> Example: <pre>clip ("cli-command")</pre>	Prints the output of the CLI command directly to stdout and returns nothing to Python.  <b>Note</b> <pre>clip ("cli-command")</pre> is equivalent to <pre>r=cli("cli-command") print r</pre>

When two or more commands are run individually, the state is not persistent from one command to subsequent commands.

In the following example, the second command fails because the state from the first command does not persist for the second command:

```
>>> cli("conf t")
>>> cli("interface eth4/1")
```

When two or more commands are run together, the state is persistent from one command to subsequent commands.

In the following example, the second command is successful because the state persists for the second and third commands:

```
>>> cli("conf t ; interface eth4/1 ; shut")
```



**Note** Commands are separated with ";" as shown in the example. (The ; must be surrounded with single blank characters.)

## Invoking the Python Interpreter from the CLI

The following example shows how to invoke Python from the CLI:



**Note** The Python interpreter is designated with the ">>>" or "... " prompt.

```
switch# python
Python 2.7.5 (default, Oct 8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 5 ; no shut')
''
>>> intflist=json.loads(cli('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...     intf=intflist['TABLE_interface']['ROW_interface'][i]
...     i=i+1
...     if intf['state'] == 'up':
...         print intf['interface']
...
mgmt0
Ethernet2/7
Ethernet4/7
loopback0
loopback5
>>>
```

## Display Formats

The following examples show various display formats using the Python APIs:

Example 1:

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> cli('where detail')
mode:
username:          admin
vdc:               switch
routing-context vrf: default
```

Example 2:

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> cli('where detail')
' mode:          \n username:          admin\n vdc:
switch\n routing-context vrf: default\n'
>>>
```

Example 3:

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
```

```

''
>>> r = cli('where detail') ; print r
mode:
username:          admin
vdc:               EOR-1
routing-context vrf: default
>>>

```

#### Example 4:

```

>>> from cli import *
>>> import json
>>> out=json.loads(cli('show version'))
>>> for k in out.keys():
...   print "%30s = %s" % (k, out[k])
...
                                kern_uptm_secs = 6
                                kick_file_name = bootflash:///n3600-dk9.6.1.2.I1.1.bin
                                rr_service = None
                                module_id = Supervisor Module
                                kick_tmstamp = 10/21/2013 00:06:10
                                bios_cmpl_time = 08/17/2013
                                bootflash_size = 20971520
                                kickstart_ver_str = 6.1(2)I1(2) [build 6.1(2)I1(2)] [gdb]
                                kick_cmpl_time = 10/20/2013 4:00:00
                                chassis_id = Nexus3600 C9508 (8 Slot) Chassis
                                proc_board_id = SAL171211LX
                                    memory = 16077872
                                manufacturer = Cisco Systems, Inc.
                                kern_uptm_mins = 26
                                bios_ver_str = 06.14
                                    cpu_name = Intel(R) Xeon(R) CPU E5-2403
                                kern_uptm_hrs = 2
                                    rr_usecs = 816550
                                    rr_sys_ver = None
                                    rr_reason = Reset Requested by CLI command reload
                                    rr_ctime = Mon Oct 21 00:10:24 2013
                                header_str = Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Documents: http://www.cisco.com/en/US/products/ps9372/tsd\_products\_support\_series\_home.html
Copyright (c) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained herein are owned by
other third parties and are used and distributed under license.
Some parts of this software are covered under the GNU Public
License. A copy of the license is available at
http://www.gnu.org/licenses/gpl.html.
                                host_name = switch
                                mem_type = kB
                                kern_uptm_days = 0
>>>

```

## Non-interactive Python

A Python script can run in non-interactive mode by providing the Python script name as an argument to the Python CLI command. Python scripts must be placed under the bootflash or volatile scheme. A maximum of 32 command line arguments for the Python script are allowed with the Python CLI command.

The Cisco Nexus 3600 device also supports the source CLI command for running Python scripts. The `bootflash:scripts` directory is the default script directory for the source CLI command.

The following example shows a script and how to run it:

```
switch# show file bootflash:deltaCounters.py
#!/isan/bin/python

from cli import *
import sys, time

ifName = sys.argv[1]
delay = float(sys.argv[2])
count = int(sys.argv[3])
cmd = 'show interface ' + ifName + ' counters'

out = json.loads(clid(cmd))
rxuc = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
rxmc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
rxbc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
txuc = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
txmc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
txbc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
print 'row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast'
print '=====
print '      %8d %8d %8d %8d %8d %8d' % (rxuc, rxmc, rxbc, txuc, txmc, txbc)
print '=====

i = 0
while (i < count):
    time.sleep(delay)
    out = json.loads(clid(cmd))
    rxucNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
    rxmcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
    rxbcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
    txucNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
    txmcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
    txbcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
    i += 1
    print '%-3d %8d %8d %8d %8d %8d %8d' % \
        (i, rxucNew - rxuc, rxmcNew - rxmc, rxbcNew - rxbc, txucNew - txuc, txmcNew - txmc,
        txbcNew - txbc)

switch# python bootflash:deltaCounters.py Ethernet1/1 1 5
row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast
=====
              0       791           1           0   212739           0
=====
1           0           0           0           0        26           0
2           0           0           0           0        27           0
3           0           1           0           0        54           0
4           0           1           0           0        55           0
5           0           1           0           0        81           0
switch#
```

The following example shows how a source command specifies command-line arguments. In the example, *policy-map* is an argument to the **cgrep python** script. The example also shows that a source command can follow after the pipe operator ("**|**").

```
switch# show running-config | source sys/cgrep policy-map

policy-map type network-qos nw-pfc
policy-map type network-qos no-drop-2
policy-map type network-qos wred-policy
policy-map type network-qos pause-policy
policy-map type qos foo
```

```

policy-map type qos classify
policy-map type qos cos-based
policy-map type qos no-drop-2
policy-map type qos pfc-tor-port

```

## Running Scripts with Embedded Event Manager

On Cisco Nexus 3600 devices, embedded event manager (EEM) policies support Python scripts.

The following example shows how to run a Python script as an EEM action:

- An EEM applet can include a Python script with an action command.

```

switch# show running-config eem

!Command: show running-config eem
!Time: Sun May  1 14:40:07 2011

version 6.1(2)I2(1)
event manager applet a1
  event cli match "show clock"
  action 1 cli python bootflash:pydate.py
  action 2 event-default

```

- You can search for the action triggered by the event in the log file by running the **show file logflash:event\_archive\_1** command.

```

switch# show file logflash:event_archive_1 | last 33

eem_event_time:05/01/2011,19:40:28 event_type:cli event_id:8 slot:active(1)
vdc:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
Python

2011-05-01 19:40:28.644891
Executing the following commands succeeded:
    python bootflash:pydate.py

PC_VSH_CMD_TLV(7679) with q

```

## Python Integration with Cisco NX-OS Network Interfaces

On Cisco Nexus 3600 devices, Python is integrated with the underlying Cisco NX-OS network interfaces. You can switch from one virtual routing context to another by setting up a context through the `cisco.vrf.set_global_vrf()` API.

The following example shows how to retrieve an HTML document over the management interface of a device. You can also establish a connection to an external entity over the inband interface by switching to a desired virtual routing context.

```

switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib2
>>> from cisco.vrf import *
>>> set_global_vrf('management')
>>> page=urllib2.urlopen('http://172.23.40.211:8000/welcome.html')

```

```

>>> print page.read()
Hello Cisco Nexus 3600

>>>
>>> import cisco
>>> help(cisco.vrf.set_global_vrf)
Help on function set_global_vrf in module cisco.vrf:

set_global_vrf(vrf)
    Sets the global vrf. Any new sockets that are created (using socket.socket)
    will automatically get set to this vrf (including sockets used by other
    python libraries).

    Arguments:
      vrf: VRF name (string) or the VRF ID (int).

    Returns: Nothing

>>>

```

## Cisco NX-OS Security with Python

Cisco NX-OS resources are protected by the Cisco NX-OS Sandbox layer of software and by the CLI role-based access control (RBAC).

All users associated with a Cisco NX-OS network-admin or dev-ops role are privileged users. Users who are granted access to Python with a custom role are regarded as non-privileged users. Non-privileged users have a limited access to Cisco NX-OS resources, such as file system, guest shell, and Bash commands. Privileged users have greater access to all the resources of Cisco NX-OS.

### Examples of Security and User Authority

The following example shows how a privileged user runs commands:

```

switch# python
Python 2.7.5 (default, Oct 8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
admin
0
>>> f=open('/tmp/test','w')
>>> f.write('hello from python')
>>> f.close()
>>> r=open('/tmp/test','r')
>>> print r.read()
hello from python
>>> r.close()

```

The following example shows a non-privileged user being denied access:

```

switch# python
Python 2.7.5 (default, Oct 8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
system(whoami): rejected!
-1
>>> f=open('/tmp/test','r')

```

```

Permission denied. Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 13] Permission denied: '/tmp/test'
>>>

```

RBAC controls CLI access based on the login user privileges. A login user's identity is given to Python that is invoked from the CLI shell or from Bash. Python passes the login user's identity to any subprocess that is invoked from Python.

The following is an example for a privileged user:

```

>>> from cli import *
>>> cli('show clock')
'11:28:53.845 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf')
''
>>> clip('show running-config l3vm')

!Command: show running-config l3vm
!Time: Sun May  8 11:29:40 2011

version 6.1(2)I2(1)

interface Ethernet1/48
  vrf member blue

interface mgmt0
  vrf member management
vrf context blue
vrf context management
vrf context myvrf

```

The following is an example for a non-privileged user:

```

>>> from cli import *
>>> cli('show clock')
'11:18:47.482 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/isan/python/scripts/cli.py", line 20, in cli
    raise cmd_exec_error(msg)
errors.cmd_exec_error: '% Permission denied for the role\n\nCmd exec error.\n'

```

The following example shows an RBAC configuration:

```

switch# show user-account
user:admin
  this user account has no expiry date
  roles:network-admin
user:pyuser
  this user account has no expiry date
  roles:network-operator python-role
switch# show role name python-role

```

## Example of Running Script with Scheduler

The following example shows a Python script that is running the script with the scheduler feature:

```

#!/bin/env python
from cli import *

```

```

from nxos import *
import os

switchname = cli("show switchname")
try:
    user = os.environ['USER']
except:
    user = "No user"
    pass

msg = user + " ran " + __file__ + " on : " + switchname
print msg
py_syslog(1, msg)
# Save this script in bootflash:///scripts

switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# feature scheduler
switch(config)# scheduler job name testplan
switch(config-job)# python bootflash:///scripts/testplan.py
switch(config-job)# exit
switch(config)# scheduler schedule name testplan
switch(config-schedule)# job name testplan
switch(config-schedule)# time start now repeat 0:0:4
Schedule starts from Mon Mar 14 16:40:03 2011
switch(config-schedule)# end
switch# term mon
2011 Mar 14 16:38:03 switch %VSHD-5-VSHD_SYSLOG_CONFIG_I: Configured from vty by admin on
10.19.68.246@pts/2
switch# show scheduler schedule
Schedule Name          : testplan
-----
User Name              : admin
Schedule Type          : Run every 0 Days 0 Hrs 4 Mins
Start Time             : Mon Mar 14 16:40:03 2011
Last Execution Time    : Yet to be executed
-----
Job Name               Last Execution Status
-----
testplan               -NA-
=====
switch#
switch# 2011 Mar 14 16:40:04 switch %USER-1-SYSTEM_MSG: No user ran
/bootflash/scripts/testplan.py on : switch - nxpython
2011 Mar 14 16:44:04 switch last message repeated 1 time
switch#

```





## CHAPTER 6

# Scripting with Tcl

- [About Tcl, on page 65](#)
- [Running the Telsh Command, on page 68](#)
- [Navigating Cisco NX-OS Modes from the Telsh Command, on page 69](#)
- [Tcl References, on page 70](#)

## About Tcl

Tcl (pronounced "tickle") is a scripting language that increases flexibility of CLI commands. You can use Tcl to extract certain values in the output of a **show** command, perform switch configurations, run Cisco NX-OS commands in a loop, or define Embedded Event Manager (EEM) policies in a script.

This section describes how to run Tcl scripts or run Tcl interactively on Cisco NX-OS devices.

## Guidelines and Limitations

Following are guidelines and limitations for TCL scripting:

- Tcl is supported on the Cisco Nexus 9508 switch.
- Some processes and **show** commands can cause a large amount of output. If you are running scripts, and need to terminate long-running output, use Ctrl+C (not Ctrl+Z) to terminate the command output. If you use Ctrl+Z, a SIGCONT (signal continuation) message can be generated, which can cause the script to halt. Scripts that are halted through SIGCONT messages require user intervention to resume operation.

## Tclsh Command Help

Command help is not available for Tcl commands. You can still access the help functions of Cisco NX-OS commands from within an interactive Tcl shell.

This example shows the lack of Tcl command help in an interactive Tcl shell:

```
switch# tclsh
switch-tcl# set x 1
switch-tcl# puts ?
      ^
% Invalid command at '^' marker.
switch-tcl# configure ?
<CR>
```

```

    session  Configure the system in a session
    terminal Configure the system from terminal input

switch-tcl#

```




---

**Note** In the preceding example, the Cisco NX-OS command help function is still available but the Tcl **puts** command returns an error from the help function.

---

## Tclsh Command History

You can use the arrow keys on your terminal to access commands you previously entered in the interactive Tcl shell.




---

**Note** The **tclsh** command history is not saved when you exit the interactive Tcl shell.

---

## Tclsh Tab Completion

You can use tab completion for Cisco NX-OS commands when you are running an interactive Tcl shell. Tab completion is not available for Tcl commands.

## Tclsh CLI Command

Although you can directly access Cisco NX-OS commands from within an interactive Tcl shell, you can only execute Cisco NX-OS commands in a Tcl script if they are prepended with the Tcl **cli** command.

In an interactive Tcl shell, the following commands are identical and execute properly:

```

switch-tcl# cli show module 1 | incl Mod
switch-tcl# cli "show module 1 | incl Mod"
switch-tcl# show module 1 | incl Mod

```

In a Tcl script, you must prepend Cisco NX-OS commands with the Tcl **cli** command as shown in the following example:

```

set x 1
cli show module $x | incl Mod
cli "show module $x | incl Mod"

```

If you use the following commands in your script, the script fails and the Tcl shell displays an error:

```

show module $x | incl Mod
"show module $x | incl Mod"

```

## Tclsh Command Separation

The semicolon (;) is the command separator in both Cisco NX-OS and Tcl. To execute multiple Cisco NX-OS commands in a Tcl command, you must enclose the Cisco NX-OS commands in quotes ("").

In an interactive Tcl shell, the following commands are identical and execute properly:

```
switch-tcl# cli "configure terminal ; interface loopback 10 ; description loop10"
switch-tcl# cli configure terminal ; cli interface loopback 10 ; cli description loop10
switch-tcl# cli configure terminal
Enter configuration commands, one per line. End with CNTL/Z.

switch(config-tcl)# cli interface loopback 10
switch(config-if-tcl)# cli description loop10
switch(config-if-tcl)#
```

In an interactive Tcl shell, you can also execute Cisco NX-OS commands directly without prepending the Tcl **cli** command:

```
switch-tcl# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.

switch(config-tcl)# interface loopback 10
switch(config-if-tcl)# description loop10
switch(config-if-tcl)#
```

## Tcl Variables

You can use Tcl variables as arguments to the Cisco NX-OS commands. You can also pass arguments into Tcl scripts. Tcl variables are not persistent.

The following example shows how to use a Tcl variable as an argument to a Cisco NX-OS command:

```
switch# tclsh
switch-tcl# set x loop10
switch-tcl# cli "configure terminal ; interface loopback 10 ; description $x"
switch(config-if-tcl)#
```

## Tclquit

The **tclquit** command exits the Tcl shell regardless of which Cisco NX-OS command mode is currently active. You can also press **Ctrl-C** to exit the Tcl shell. The **exit** and **end** commands change Cisco NX-OS command modes. The **exit** command terminates the Tcl shell only from the EXEC command mode.

## Tclsh Security

The Tcl shell is executed in a sandbox to prevent unauthorized access to certain parts of the Cisco NX-OS system. The system monitors CPU, memory, and file system resources being used by the Tcl shell to detect events such as infinite loops, excessive memory utilization, and so on.

You configure the initial Tcl environment with the **scripting tcl init** *init-file* command.

You can define the looping limits for the Tcl environment with the **scripting tcl recursion-limit** *iterations* command. The default recursion limit is 1000 iterations.

# Running the Tclsh Command

You can run Tcl commands from either a script or on the command line using the **tclsh** command.



**Note** You cannot create a Tcl script file at the CLI prompt. You can create the script file on a remote device and copy it to the bootflash: directory on the Cisco NX-OS device.

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	<pre>tclsh [bootflash:filename [argument ... ]]</pre> <p><b>Example:</b></p> <pre>switch# tclsh ? &lt;CR&gt; bootflash: The file to run</pre>	<p>Starts a Tcl shell.</p> <p>If you run the <b>tclsh</b> command with no arguments, the shell runs interactively, reading Tcl commands from standard input and printing command results and error messages to the standard output. You exit from the interactive Tcl shell by typing <b>tclquit</b> or <b>Ctrl-C</b>.</p> <p>If you run the <b>tclsh</b> command with arguments, the first argument is the name of a script file containing Tcl commands and any additional arguments are made available to the script as variables.</p>

## Example

The following example shows an interactive Tcl shell:

```
switch# tclsh
switch-tcl# set x 1
switch-tcl# cli show module $x | incl Mod
Mod  Ports  Module-Type                Model                Status
1    36      36p 40G Ethernet Module    N9k-X9636PQ         ok
Mod  Sw      Hw
Mod  MAC-Address(es)          Serial-Num
```

```
switch-tcl# exit
switch#
```

The following example shows how to run a Tcl script:

```
switch# show file bootflash:showmodule.tcl
set x 1
while {$x < 19} {
cli show module $x | incl Mod
set x [expr {$x + 1}]
}

switch# tclsh bootflash:showmodule.tcl
Mod  Ports  Module-Type                Model                Status
```

```

1    36    36p 40G Ethernet Module          N9k-X9636PQ    ok
Mod  Sw              Hw
Mod  MAC-Address(es)          Serial-Num

switch#

```

## Navigating Cisco NX-OS Modes from the Tclsh Command

You can change modes in Cisco NX-OS while you are running an interactive Tcl shell.

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>tclsh</b> <b>Example:</b> switch# <b>tclsh</b> switch-tcl#	Starts an interactive Tcl shell.
<b>Step 2</b>	<b>configure terminal</b> <b>Example:</b> switch-tcl# <b>configure terminal</b> switch(config-tcl)#	Runs a Cisco NX-OS command in the Tcl shell, changing modes.  <b>Note</b> The Tcl prompt changes to indicate the Cisco NX-OS command mode.
<b>Step 3</b>	<b>tclquit</b> <b>Example:</b> switch-tcl# <b>tclquit</b> switch#	Terminates the Tcl shell, returning to the starting mode.

### Example

The following example shows how to change Cisco NX-OS modes from an interactive Tcl shell:

```

switch# tclsh
switch-tcl# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config-tcl)# interface loopback 10
switch(config-if-tcl)# ?
  description  Enter description of maximum 80 characters
  inherit      Inherit a port-profile
  ip           Configure IP features
  ipv6         Configure IPv6 features
  logging      Configure logging for interface
  no           Negate a command or set its defaults
  rate-limit   Set packet per second rate limit
  shutdown     Enable/disable an interface
  this         Shows info about current object (mode's instance)
  vrf          Configure VRF parameters
  end         Go to exec mode

```

```
exit          Exit from command interpreter
pop           Pop mode from stack or restore from name
push         Push current mode to stack or save it under name
where        Shows the cli context you are in

switch(config-if-tcl)# description loop10
switch(config-if-tcl)# tclquit
Exiting Tcl
switch#
```

## Tcl References

The following titles are provided for your reference:

- Mark Harrison (ed), *Tcl/Tk Tools*, O'Reilly Media, ISBN 1-56592-218-2, 1997
- Mark Harrison and Michael McLennan, *Effective Tcl/Tk Programming*, Addison-Wesley, Reading, MA, USA, ISBN 0-201-63474-0, 1998
- John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, Reading, MA, USA, ISBN 0-201-63337-X, 1994.
- Brent B. Welch, *Practical Programming in Tcl and Tk*, Prentice Hall, Upper Saddle River, NJ, USA, ISBN 0-13-038560-3, 2003.
- J Adrian Zimmer, *Tcl/Tk for Programmers*, IEEE Computer Society, distributed by John Wiley and Sons, ISBN 0-8186-8515-8, 1998.



## CHAPTER 7

# iPXE

---

This chapter contains the following sections:

- [About iPXE, on page 71](#)
- [Netboot Requirements, on page 72](#)
- [Guidelines and Limitations, on page 72](#)
- [Boot Mode Configuration, on page 80](#)
- [Verifying the Boot Order Configuration, on page 82](#)

## About iPXE

iPXE is an open source network boot firmware. iPXE is based on gPXE, which is an open-source PXE client firmware and bootloader derived from Etherboot. Standard PXE clients use TFTP to transfer data, whereas gPXE supports additional protocols.

Here is a list of additional features that iPXE provides over standard PXE:

- Boots from a web server via HTTP, iSCSI SAN, FCoE, etc.,
- Supports both IPv4 and IPv6,
- Netboot supports HTTP/TFTP, IPv4, and IPv6,
- Supports embedded scripts into the image or served by the HTTP/TFTP, etc., and
- Supports stateless address auto-configuration (SLAAC) and stateful IP auto-configuration variants for DHCPv6. iPXE supports boot URI and parameters for DHCPv6 options. This depends on IPv6 router advertisement.

In addition, we have disabled some of the existing features from iPXE for security reasons such as:

- Boot support for standard Linux image format such as bzImage+initramfs/initrd, or ISO, etc.,
- Unused network boot options such as FCoE, iSCSI SAN, Wireless, etc., and
- Loading of unsupported NBP (such as syslinux/pxelinux) because these might boot system images that are not properly code-signed.

# Netboot Requirements

The primary requirements are:

- A DHCP server with proper configuration.
- A TFTP/HTTP server.
- Enough space on the device's bootflash because NX-OS downloads the image when the device is PXE booted.
- IPv4/IPv6 support—for better deployment flexibility

## Guidelines and Limitations

PXE has the following configuration guidelines and limitations:

- While auto-booting through iPXE, there is a window of three seconds where you can enter **Ctrl+B** to exit out of the PXE boot. The system prompts you with the following options:

```
Please choose a bootloader shell:
1). GRUB shell
2). PXE shell
Enter your choice:
```

- HTTP image download vs. TFTP—TFTP is UDP based and it can be problematic if packet loss starts appearing. TCP is a window-based protocol and handles bandwidth sharing/losses better. As a result, TCP-based protocols support is more suitable given the sizes of the Cisco Nexus images which are over 250 Mbytes.
- iPXE only allows/boots Cisco signed NBI images. Other standard image format support is disabled for security reasons.

## Notes for iPXE

### DHCP server installation

DHCP is not installed in the server by default. You can verify DHCP server installation with the **service dhcpd status** command.

```
[switch etc]# service dhcpd status
dhcpd: unrecognized service /* indicates that dhcp server is not installed */
```

You can install DHCP with the **yum install dhcp** command.



---

**Note** Root credentials are required for installing the DHCP server.

---



```
[switch etc]# yum install dhcp
Repository base is listed more than once in the configuration
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package dhcp.x86_64 12:3.0.5-23.e15 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved
=====

Package Arch Version Repository
Size
=====
Installing:
dhcp x86_64 12:3.0.5-23.e15 workstation 883
k

Transaction Summary
=====
Install 1 Package(s)
Upgrade 0 Package(s)

Total download size: 883 k
Is this ok [y/N]: y
Downloading Packages:
dhcp-3.0.5-23.e15.x86_64.rpm | 883 kB 00:00
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing      : dhcp
1/1
Installed:
  dhcp.x86_64 12:3.0.5-23.e15

Complete!
[switch etc]#
```

### Adding a configuration to the DHCP server

After the DHCP server is installed, the configuration file is located at `/etc/dhcpd.conf`.

The following is an example of the `dhcpd.conf` file.

```
-----
# Set the amount of time in seconds that a client may keep the IP address
default-lease-time 300;
max-lease-time 7200;
one-lease-per-client true;

#Indicate the preferred interface that your DHCP server listens only to that interface and
to no other . Preferred interface should be added to the DHCPDARGS variable
DHCPDARGS=eth0

#A subnet section is generated for each of the interfaces present on your Linux system
subnet 10.0.00.0 netmask 255.255.255.0 {

# The range of IP addresses the server will issue to DHCP enabled PC clients booting up on
the network
```

```

range 10.0.00.2 10.0.00.100;

#Address of the preferred interface
next-server 10.0.00.4;

#The default gateway to be used
option routers 10.0.00.254;

#The file path where the ipxe boot looks for the image
filename = "http://10.0.00.4/pxe/dummy";
# (http://10.0.00.4 points to the httpd service path mentioned in DocumentRoot variable
# at /etc/httpd/conf/httpd.conf ) .
# By default it points to "DocumentRoot "/var/www/html" (Refer the HTTP service section)

option domain-name "cisco.com";
option domain-name-servers 100.00.000.183;

host Nexus {
    hardware ethernet e4:c7:22:bd:c4:f9;
    fixed-address 10.0.00.42;
    filename = "http://10.0.00.4/ipxe/nxos-image.bin";

host Nexus {
    hardware ethernet 64:f6:9d:07:52:f7;
    fixed-address 10.0.00.8;
    filename = "tftp://100.00.000.48/nxos-image.bin";
-----

```

## Managing the DHCP service




---

**Note** After installing the DHCP service, you need to initiate the service.

---

- Verifying the DHCP service

```
[switch etc]# service dhcpd status
dhcpd is stopped
```

- Starting the DHCP service

```
[switch etc]# service dhcpd start
Starting dhcpd: [ok]
```

- Stopping the DHCP service

```
[switch etc]# service dhcpd stop
Stopping dhcpd: [ok]
```

- Restarting the DHCP service




---

**Note** When the DHCP configuration file **/etc/dhcpd.conf** is updated, you need to restart the service.

---

```
[switch etc]# service dhcpd restart
Starting dhcpd: [ok]
```

## Managing the HTTP server

- HTTP server installation

```
[switch conf]# yum install httpd
```

- Starting the HTTP service

```
[switch conf]# service httpd start
Starting httpd: httpd: Could not reliably determine the server's fully qualified domain
name,
using 100.00.000.127 for ServerName
[ OK ]
```

- Stopping the HTTP service

```
[switch conf]# service httpd stop
Stopping httpd: [ OK ]
```

- Restarting the HTTP service

```
[switch conf]# service httpd restart
Stopping httpd: [FAILED]
Starting httpd: httpd: Could not reliably determine the server's fully qualified domain
name,
using 100.00.000.127 for ServerName
[ OK ]
```

- Verifying the HTTP status

```
[switch conf]# service httpd status
httpd (pid 23032) is running...
```



---

**Note** The HTTP configuration file is located at `/etc/httpd/conf/httpd.conf`.

---

**Note**

- DocumentRoot: The directory out of which you will serve your documents. By default, all requests are taken from this directory, but symbolic links and aliases may be used to point to other locations.

- DocumentRoot **/var/www/html**

The DocumentRoot variable contains the path that represents the `http://<ip_add>` field in the **dhcpd.conf** file with the filename variable.

The following is an example:

```
host Nexus {
    hardware ethernet e4:c7:22:bd:c4:f9;
    fixed-address 10.0.0.42;
    filename = "http://10.0.0.4/ipxe/nxos-image.bin";
```

The filename path redirects to the location **/var/www/html/ipxe/nxos-image.bin**, where the ipxe bootup looks for the image.

- TFTP server installation

```
[switch conf]# yum install tftp
```

The TFTP configuration file located at **/etc/xinetd.d/tftp**.

The following is an example of a TFTP configuration file:

```
[switch xinetd.d]# cat tftp
# default: off
# description: The tftp server serves files using the trivial file transfer \
#               protocol. The tftp protocol is often used to boot diskless \
#               workstations, download configuration files to network-aware printers, \
#               and to start the installation process for some operating systems.
service tftp
{
    disable = no
    socket_type      = dgram
    protocol        = udp
    wait            = yes
    user            = root
    server          = /usr/sbin/in.tftpd
    server_args     = -s /tftpboot # Indicates the tftp path
    per_source     = 11
    cps             = 100 2
    flags           = IPv4
}
```

- Stopping the TFTP service

```
[switch xinetd.d]# chkconfig tftp off
```

- Starting the TFTP service

```
[switch xinetd.d]# chkconfig tftp on
```




---

**Note** When you change the TFTP configuration file, you need to restart the TFTP service.

---

```
host Nexus {
    hardware ethernet 64:f6:9d:07:52:f7;
    fixed-address 10.0.00.8;
    filename = "tftp://100.00.000.48/nxos-image.bin";
}
```




---

**Note** A prerequisite is that the nxos\_image.bin has to be copied to /tftpboot shown in the above example TFTP path /tftpboot.

---

- iPXE using HTTP protocol

```
Nexus# sh int mgmt0
mgmt0 is up
admin state is up,
  Hardware: GigabitEthernet, address: e4c7.22bd.c4a6 (bia e4c7.22bd.c4a6)
  Internet Address is 10.0.00.42/24
  MTU 1500 bytes, BW 100000 Kbit, DLY 10 usec
  reliability 255/255, txload 1/255, rxload 1/255
Encapsulation ARPA, medium is broadcast
full-duplex, 100 Mb/s
Auto-Negotiation is turned on
Auto-mdix is turned off
EtherType is 0x0000
1 minute input rate 312 bits/sec, 0 packets/sec
1 minute output rate 24 bits/sec, 0 packets/sec
Rx
  5433 input packets 10 unicast packets 5368 multicast packets
  55 broadcast packets 405677 bytes
Tx
  187 output packets 9 unicast packets 175 multicast packets
  3 broadcast packets 45869 bytes
Nexus#

Nexus# ping 199.00.000.48 vrf management
PING 199.00.000.48 (199.00.000.48): 56 data bytes
64 bytes from 199.00.000.48: icmp_seq=0 ttl=61 time=82.075 ms
64 bytes from 199.00.000.48: icmp_seq=1 ttl=61 time=0.937 ms
64 bytes from 199.00.000.48: icmp_seq=2 ttl=61 time=0.861 ms
64 bytes from 199.00.000.48: icmp_seq=3 ttl=61 time=0.948 ms
64 bytes from 199.00.000.48: icmp_seq=4 ttl=61 time=0.961 ms

--- 199.00.000.48 ping statistics ---
5 packets transmitted, 5 packets received, 0.00% packet loss
round-trip min/avg/max = 0.861/17.156/82.075 ms
Nexus# conf t
Enter configuration commands, one per line. End with CNTL/Z.
Nexus(config)# no boot nxos
Nexus(config)# boot order pxe bootflash
Nexus(config)# end

Nexus# copy running-config startup-config
[#####] 100%
Copy complete, now saving to disk (please wait)...
```

```

Copy complete.
Nexus# reload
This command will reboot the system. (y/n)? [n] y

CISCO SWITCH Ver 8.32

CISCO SWITCH Ver 8.32
Memory Size (Bytes): 0x0000000080000000 + 0x00000000380000000
Relocated to memory
Time: 9/8/2017 1:3:28
Detected CISCO IOFPGA
Booting from Primary Bios
Code Signing Results: 0x0
Using Upgrade FPGA
FPGA Revision      : 0x20
FPGA ID            : 0x1168153
FPGA Date          : 0x20140317
Reset Cause Register: 0x20
Boot Ctrl Register : 0x60ff
EventLog Register1 : 0xc2004000
EventLog Register2 : 0xfbc77fff
Version 2.16.1240. Copyright (C) 2013 American Megatrends, Inc.
Board type 1
IOFPGA @ 0xe8000000
SLOT_ID @ 0x1b
Standalone chassis
check_bootmode: pxe2grub: Launch pxe
Trying to load ipxe
Loading Application:
/Vendor(429bdb26-48a6-47bd-664c-801204061400)/UnknownMedia(6)/EndEntire
iPXE initialising devices...ok

Cisco iPXE
iPXE 1.0.0+ (3cb3) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: HTTP DNS TFTP NBI Menu
net6: e4:c7:22:bd:c4:a6 using dh8900cc on PCI02:00.3 (open)
[Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net6 e4:c7:22:bd:c4:a6)..... ok
net0: fe80::2a0:c9ff:fe00:0/64 (inaccessible)
net1: fe80::2a0:c9ff:fe00:1/64 (inaccessible)
net2: fe80::2a0:c9ff:fe00:2/64 (inaccessible)
net3: fe80::2a0:c9ff:fe00:3/64 (inaccessible)
net4: fe80::200:ff:fe00:5/64 (inaccessible)
net5: fe80::200:ff:fe00:7/64 (inaccessible)
net6: 10.0.00.7/255.255.255.0 gw 10.0.00.254
net6: fe80::e6c7:22ff:febd:c4a5/64
net7: fe80::200:ff:fe00:0/64 (inaccessible)
Next server: 10.0.00.4
Filename: http://10.0.00.4/ipxe/nxos-image.bin
http://10.0.00.4/ipxe/nxos-image.bin... ok
http://10.0.00.4/ipxe/nxos_image.bin... 46%
Further device bootsup fine .

```

- iPXE using TFTP protocol

```

nexus# sh int mgmt0
mgmt0 is up
admin state is up,
  Hardware: GigabitEthernet, address: e4c7.22bd.c4a6 (bia e4c7.22bd.c4a6)
  Internet Address is 10.0.00.8/24
  MTU 1500 bytes, BW 100000 Kbit, DLY 10 usec
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, medium is broadcast

```

```
full-duplex, 100 Mb/s
Auto-Negotiation is turned on
Auto-mdix is turned off
EtherType is 0x0000
1 minute input rate 312 bits/sec, 0 packets/sec
1 minute output rate 24 bits/sec, 0 packets/sec
Rx
  5433 input packets 10 unicast packets 5368 multicast packets
  55 broadcast packets 405677 bytes
Tx
  187 output packets 9 unicast packets 175 multicast packets
  3 broadcast packets 45869 bytes
nexus#
nexus# ping 199.00.000.48 vrf management
PING 199.00.000.48 (199.00.000.48): 56 data bytes
64 bytes from 199.00.000.48: icmp_seq=0 ttl=61 time=82.075 ms
64 bytes from 199.00.000.48: icmp_seq=1 ttl=61 time=0.937 ms
64 bytes from 199.00.000.48: icmp_seq=2 ttl=61 time=0.861 ms
64 bytes from 199.00.000.48: icmp_seq=3 ttl=61 time=0.948 ms
64 bytes from 199.00.000.48: icmp_seq=4 ttl=61 time=0.961 ms

--- 199.00.000.48 ping statistics ---
5 packets transmitted, 5 packets received, 0.00% packet loss
round-trip min/avg/max = 0.861/17.156/82.075 ms

nexus# conf t
Enter configuration commands, one per line. End with CNTL/Z.
nexus(config)# no boot nxos
nexus(config)# boot order pxe bootflash
nexus(config)# end

nexus# copy running-config startup-config
[#####] 100%
Copy complete, now saving to disk (please wait)...
Copy complete.

nexus# reload
This command will reboot the system. (y/n)? [n] y

CISCO SWITCH Ver 8.32

CISCO SWITCH Ver 8.32
Memory Size (Bytes): 0x0000000080000000 + 0x00000000380000000
Relocated to memory
Time: 9/8/2017 1:3:28
Detected CISCO IOFPGA
Booting from Primary Bios
Code Signing Results: 0x0
Using Upgrade FPGA
FPGA Revision      : 0x20
FPGA ID            : 0x1168153
FPGA Date          : 0x20140317
Reset Cause Register: 0x20
Boot Ctrl Register : 0x60ff
EventLog Register1 : 0xc2004000
EventLog Register2 : 0xfbc77fff
Version 2.16.1240. Copyright (C) 2013 American Megatrends, Inc.
Board type 1
IOFPGA @ 0xe8000000
SLOT_ID @ 0x1b
Standalone chassis
check_bootmode: pxe2grub: Launch pxe
Trying to load ipxe
Loading Application:
```

```

/Vendor(429bdb26-48a6-47bd-664c-801204061400)/UnknownMedia(6)/EndEntire
iPXE initialising devices...ok

Cisco iPXE
iPXE 1.0.0+ (3cb3) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: HTTP DNS TFTP NBI Menu
net6: e4:c7:22:bd:c4:a6 using dh8900cc on PCI02:00.3 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net6 e4:c7:22:bd:c4:a6)..... ok
net0: fe80::2a0:c9ff:fe00:0/64 (inaccessible)
net1: fe80::2a0:c9ff:fe00:1/64 (inaccessible)
net2: fe80::2a0:c9ff:fe00:2/64 (inaccessible)
net3: fe80::2a0:c9ff:fe00:3/64 (inaccessible)
net4: fe80::200:ff:fe00:5/64 (inaccessible)
net5: fe80::200:ff:fe00:7/64 (inaccessible)
net6: 10.0.00.7/255.255.255.0 gw 10.0.00.254
net6: fe80::e6c7:22ff:febd:c4a5/64
net7: fe80::200:ff:fe00:0/64 (inaccessible)
Next server: 10.0.00.4
filename: tftp://199.00.000.48/nxos-image.bin
tftp://199.00.000.48/nxos-image.bin... ok
tftp://199.00.000.48/nxos_image.bin... 26%

*****

```

- Interrupting the process

Use **ctrl-B** to interrupt the process and reach the iPXE shell.

- The following is an example of booting an image residing on the PXE server using HTTP protocol:

```

iPXE> dhcp
Configuring (net6 e4:c7:22:bd:c4:a6)..... ok
iPXE>boot http://10.0.0.4/ipxe/nxos-image.bin

```

- The following is an example of booting an image residing on the PXE server using TFTP protocol:

```

iPXE> dhcp
iPXE> boot tftp://199.00.00.48/nxos-image.bin

```

Use **exit** to exit the iPXE shell.

## Boot Mode Configuration

### VSH CLI

```

switch# configure terminal
switch(conf)# boot order bootflash|pxe [bootflash|pxe]
switch(conf)# end

```




---

**Note** The keyword **bootflash** indicates it is Grub based booting.

---



For example, to do a PXE boot mode only, the configuration command is:

```
switch(conf)# boot order pxe
```

To boot Grub first, followed by PXE:

```
switch(conf)# boot order bootflash pxe
```

To boot PXE first, followed by Grub:

```
switch(conf)# boot order pxe bootflash
```

If you never use the **boot order** command, by default the boot order is Grub.



**Note** The following sections describe how you can toggle from Grub and iPXE.

### Grub CLI

```
bootmode [-g|-p|-p2g|-g2p]
```

Keyword	Function
<b>-g</b>	Grub only
<b>-p</b>	PXE only
<b>-p2g</b>	PXE first, followed by Grub if PXE failed
<b>-g2p</b>	Grub first, followed by PXE if Grub failed

The Grub CLI is useful if you want to toggle the boot mode from the serial console without booting a full Nexus image. It can also be used to get a box out of the continuous PXE boot state.

### iPXE CLI

```
bootmode [-g|--grub] [-p|--pxe] [-a|--pxe2grub] [-b|--grub2pxe]
```

Keyword	Function
<b>-- grub</b>	Grub only
<b>-- pxe</b>	PXE only
<b>-- pxe2grub</b>	PXE first, followed by Grub if PXE failed
<b>-- grub2pxe</b>	Grub first, followed by PXE if Grub failed

The iPXE CLI is useful if you wish to toggle the boot mode from the serial console without booting a full Nexus image. It can also be used to get a box out of continuous PXE boot state.

## Verifying the Boot Order Configuration

To display boot order configuration information, enter the following command:

Command	Purpose
show boot order	Displays the current boot order from the running configuration and the boot order value on the next reload from the startup configuration.



## CHAPTER 8

# Kernel Stack

---

This chapter contains the following sections:

- [About Kernel Stack, on page 83](#)
- [Guidelines and Limitations, on page 83](#)
- [Changing the Port Range, on page 84](#)

## About Kernel Stack

Kernel Stack (kstack) uses well known Linux APIs to manage the routes and front panel ports.

Open Containers, like the Guest Shell, are Linux environments that are decoupled from the host software. The customers may install or modify software within that environment without impacting the host software packages.

Kernel Stack has the following features:

## Guidelines and Limitations

Using the Kernel Stack has the following guidelines and limitations:

- Guest Shell, other open containers, and the host Bash Shell use Kernel Stack (kstack).
- Open containers start in the host default namespace
  - Other network namespaces might be accessed by using the **setns** system call
  - The **nsenter** and **ip netns exec** utilities can be used to execute within the context of a different network namespace.
  - The PIDs and identify options for the **ip netns** command do not work without modification because of the file system device check. A **vrinfo** utility is provided to give the network administrator the same information.
- Open containers may read the interface state from `/proc/net/dev` or use other normal Linux utilities such as **netstat** or **ifconfig** without modification. This provides counters for packets that have initiated / terminated on the switch.

- Open containers may use **ethtool -S** to get extended statistics from the net devices. This includes packets switched through the interface.
- Open containers may run packet capture applications like **tcpdump** to capture packets initiated from or terminated on the switch.
- There is no support for networking state changes (interface creation/deletion, IP address configuration, MTU change, etc.) from the Open containers
- IPv4 and IPv6 are supported
- Raw PF\_PACKET is supported
- Well-known ports (0-15000) may only be used by one stack (Netstack or kstack) at a time, regardless of the network namespace.
- There is no IP connectivity between Netstack and kstack applications. This is a host limitation which also applies to open containers.
- Open containers are not allowed to send packets directly over an Ethernet out-of-band channel (EOBC) interface to communicate with the linecards or standby Sup.
- From within an open container, direct access to the EOBC interface used for internal communication with linecards or the standby supervisor. The host bash shell should be used if this access is needed.
- The management interface (mgmt0) is represented as eth1 in the kernel netdevices.
- Use of the VXLAN overlay interface (NVE x) is not supported for applications utilizing the kernel stack. NX-OS features, including CLI commands, are able to use this interface via netstack.

## Changing the Port Range

Netstack and kstack divide the port range between them. The default port ranges are as follows:

- Kstack—15001 to 58000
- Netstack—58001 to 65535




---

**Note** Within this range 63536 to 65535 are reserved for NAT.

---

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<code>[no] sockets local-port-range start-port end-port</code>	This command modifies the port range for kstack. This command does not modify the Netstack range.

**Example**

The following example sets the kstack port range:

```
switch# sockets local-port-range 15001 25000
```

**What to do next**

After you have entered the command, you need to be aware of the following issues:

- You must reload the switch after entering the command.
- You must leave a minimum of 7000 ports unallocated which are used by Netstack.
- You must specify the *start-port* as 15001 or the *end-port* as 65535 to avoid holes in the port range.





## PART II

# Applications

- [Third-Party Applications, on page 89](#)
- [Ansible, on page 105](#)
- [Puppet Agent, on page 107](#)
- [Using Chef Client with Cisco NX-OS, on page 111](#)
- [Nexus Application Development - ISO, on page 115](#)
- [Nexus Application Development - SDK, on page 119](#)
- [NX-SDK, on page 127](#)
- [Using Docker with Cisco NX-OS, on page 135](#)







## CHAPTER 9

# Third-Party Applications

---

This chapter contains the following sections:

- [About Third-Party Applications](#), on page 89
- [Installing Signed Third-Party RPMs by Importing Keys Automatically](#), on page 89
- [Installing Signed RPM](#), on page 91
- [Persistent Third-Party RPMs](#), on page 96
- [Installing RPM from VSH](#), on page 97
- [Third-Party Applications](#), on page 101

## About Third-Party Applications

The RPMs for the Third-Party Applications are available in the repository at [https://devhub.cisco.com/artifactory/open-nxos/7.0-3-12-1/x86\\_64](https://devhub.cisco.com/artifactory/open-nxos/7.0-3-12-1/x86_64). These applications are installed in the native host by using the **yum** command in the Bash shell or through the NX-OS CLI.

When you enter the **yum install rpm** command, a Cisco **YUM** plugin gets executed. This plugin copies the RPM to a hidden location. On switch reload, the system re-installs the RPM.

For configurations in `/etc`, a Linux process, **incron**, monitors artifacts created in the directory and copies them to a hidden location, which gets copied back to `/etc`.

## Installing Signed Third-Party RPMs by Importing Keys Automatically

Setup the yum repo to point to the keys and RPM.

```
root@switch# cat /etc/yum/repos.d/puppet.repo
[puppet]
name=Puppet RPM
baseurl=file:///bootflash/puppet
enabled=1
gpgcheck=1
```

```

gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
metadata_expire=0
cost=500

bash-4.2# yum install puppet-enterprise

Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages

groups-repo                | 1.1 kB    00:00 ...
localdb                    | 951 B     00:00 ...
patching                   | 951 B     00:00 ...
puppet                    | 951 B     00:00 ...
thirdparty                 | 951 B     00:00 ...

Setting up Install Process

Resolving Dependencies

--> Running transaction check
---> Package puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====

Package                Arch      Version                               Repository      Size
=====
Installing:
puppet-enterprise      x86_64    3.7.1.rc2.6.g6cdc186-1.pe.nxos      puppet          14 M
Transaction Summary

=====

Install      1 Package
Total download size: 14 M
Installed size: 46 M

Is this ok [y/N]: y

Retrieving key from file:///bootflash/RPM-GPG-KEY-puppetlabs

Importing GPG key 0x4BD6EC30:

Userid: "Puppet Labs Release Key (Puppet Labs Release Key) <info@puppetlabs.com>"

```

```

From : /bootflash/RPM-GPG-KEY-puppetlabs
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Warning! Standby is not ready. This can cause RPM database inconsistency.
If you are certain that standby is not booting up right now, you may proceed.
Do you wish to continue?
Is this ok [y/N]: y
Warning: RPMDB altered outside of yum.
Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64          1/1
/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link
Installed:
puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos

Complete!

```

## Installing Signed RPM

### Checking a Signed RPM

Run the following command to check if a given RPM is signed or not.

```
Run, rpm -K rpm_file_name
```

#### Not a signed RPM

```
bash-4.2# rpm -K bgp-1.0.0-r0.lib32_n3600.rpm
bgp-1.0.0-r0.lib32_n3600.rpm: (sha1) dsa sha1 md5 OK
```

#### Signed RPM

```
bash-4.2#
rpm -K puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm: RSA sha1 MD5 NOT_OK
```

```
bash-4.2#
```

Signed third-party rpm requires public GPG key to be imported first before the package can be installed otherwise **yum** will throw the following error:

```
bash-4.2#
```

```
yum install puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm -q
```

```
Setting up Install Process
```

```
warning: rpmts_HdrFromFdno: Header V4 RSA/SHA1 signature: NOKEY, key ID 4bd6ec30
```

```
Cannot open: puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm. Skipping.
```

```
Error: Nothing to do
```

## Installing Signed RPMs by Manually Importing Key

- Copy the GPG keys to `/etc rootfs` so that they are persisted across reboots.

```
bash-4.2# mkdir -p /etc/pki/rpm-gpg
```

```
bash-4.2# cp -f RPM-GPG-KEY-puppetlabs /etc/pki/rpm-gpg/
```

- Import the keys using the below command

```
bash-4.2# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
```

```
bash-4.2#
```

```
bash-4.2# rpm -q gpg-pubkey
```

```
gpg-pubkey-4bd6ec30-4c37bb40
```

```
bash-4.2# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-puppetlabs
```

```
bash-4.2#
```

```
bash-4.2# rpm -q gpg-pubkey
```

```
gpg-pubkey-4bd6ec30-4c37bb40
```

- Install the signed RPM with *yum* command

```
bash-4.2#
```

```
yum install puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm
```

```
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching, protect-packages
```

```
groups-repo          | 1.1 kB    00:00 ...
```

```
localdb              | 951 B    00:00 ...
```

```
patching             | 951 B    00:00 ...
```

```
thirdparty           | 951 B    00:00 ...
```

Setting up Install Process

Examining puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86\_64.rpm:  
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86\_64

Marking puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86\_64.rpm to be installed

Resolving Dependencies

--> Running transaction check

---> Package puppet-enterprise.x86\_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be installed

--> Finished Dependency ResolutionDependencies Resolved

```
=====
Package           Arch    Version                               Repository
Size
```

Installing:

```
puppet-enterprise  x86_64  3.7.1.rc2.6.g6cdc186-1.pe.nxos  /puppet-enterprise-
46 M                                                    3.7.1.rc2.6.g6cdc186-1.
pe.nxos.x86_64
```

Transaction Summary

```
=====
Install           1 Package
```

Total size: 46 M

Installed size: 46 M

Is this ok [y/N]: y

Downloading Packages:

Running Transaction Check

Running Transaction Test

Transaction Test Succeeded

Running Transaction

Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86\_64

1/1

Installed:

puppet-enterprise.x86\_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos

```
Complete!
```

```
bash-4.2#
```

## Installing Signed Third-Party RPMs by Importing Keys Automatically

Setup the yum repo to point to the keys and RPM.

```
root@switch# cat /etc/yum/repos.d/puppet.repo
```

```
[puppet]
```

```
name=Puppet RPM
```

```
baseurl=file:///bootflash/puppet
```

```
enabled=1
```

```
gpgcheck=1
```

```
gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
```

```
metadata_expire=0
```

```
cost=500
```

```
bash-4.2# yum install puppet-enterprise
```

```
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
protect-packages
```

```
groups-repo                | 1.1 kB    00:00 ...
localdb                    | 951 B     00:00 ...
patching                   | 951 B     00:00 ...
puppet                     | 951 B     00:00 ...
thirdparty                 | 951 B     00:00 ...
```

```
Setting up Install Process
```

```
Resolving Dependencies
```

```
--> Running transaction check
```

```
---> Package puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos will be installed
```

```
--> Finished Dependency Resolution
```

```
Dependencies Resolved
```

```
=====
```

Package	Arch	Version	Repository	Size
=====				

```
Installing:
puppet-enterprise    x86_64    3.7.1.rc2.6.g6cdc186-1.pe.nxos    puppet    14 M

Transaction Summary
=====

Install            1 Package
Total download size: 14 M
Installed size: 46 M
Is this ok [y/N]: y
Retrieving key from file:///bootflash/RPM-GPG-KEY-puppetlabs
Importing GPG key 0x4BD6EC30:
  Userid: "Puppet Labs Release Key (Puppet Labs Release Key) <info@puppetlabs.com>"
  From   : /bootflash/RPM-GPG-KEY-puppetlabs
Is this ok [y/N]: y
Downloading Packages:
Running Transaction Check
Running Transaction Test
Transaction Test Succeeded
Running Transaction
Warning! Standby is not ready. This can cause RPM database inconsistency.
If you are certain that standby is not booting up right now, you may proceed.
Do you wish to continue?
Is this ok [y/N]: y
Warning: RPMDB altered outside of yum.
Installing : puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64    1/1

/sbin/ldconfig: /usr/lib/libboost_regex.so.1.49.0 is not a symbolic link
Installed:
puppet-enterprise.x86_64 0:3.7.1.rc2.6.g6cdc186-1.pe.nxos

Complete!
```

## Adding Signed RPM into Repo

### Procedure

#### Step 1 Copy signed RPM to repo directory

#### Step 2 Import the corresponding key for the create repo to succeed

```
bash-4.2# ls
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm RPM-GPG-KEY-puppetlabs
bash-4.2#
bash-4.2# rpm --import RPM-GPG-KEY-puppetlabs
bash-4.2# createrepo .
1/1 - puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm
Saving Primary metadata
Saving file lists metadata
Saving other metadata
bash-4.2#
```

#### Without importing keys

```
bash-4.2# ls
puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm RPM-GPG-KEY-puppetlabs
bash-4.2#
bash-4.2# createrepo .
warning: rpmts_HdrFromFdno: Header V4 RSA/SHA1 signature: NOKEY, key ID 4bd6ec30

Error opening package - puppet-enterprise-3.7.1.rc2.6.g6cdc186-1.pe.nxos.x86_64.rpm

Saving Primary metadata
Saving file lists metadata
Saving other metadata
```

#### Step 3 Create repo config file under `/etc/yum/repos.d` pointing to this repo

```
bash-4.2# cat /etc/yum/repos.d/puppet.repo
[puppet]
name=Puppet RPM
baseurl=file:///bootflash/puppet
enabled=1
gpgcheck=1
gpgkey=file:///bootflash/puppet/RPM-GPG-KEY-puppetlabs
#gpgkey=http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
metadata_expire=0
cost=500

bash-4.2# yum list available puppet-enterprise -q
Available Packages
puppet-enterprise.x86_64          3.7.1.rc2.6.g6cdc186-1.pe.nxos
                                puppet
bash-4.2#
```

## Persistent Third-Party RPMs

The following is the logic behind persistent third-party RPMs:



- A local **yum** repository is dedicated to persistent third-party RPMs. The `/etc/yum/repos.d/thirdparty.repo` points to `/bootflash/.rpmstore/thirdparty`.
- Whenever you enter the **yum install third-party.rpm** command, a copy of the RPM is saved in `//bootflash/.rpmstore/thirdparty`.
- During a reboot, all the RPMs in the third-party repository are reinstalled on the switch.
- Any change in the `/etc` configuration files persists under `/bootflash/.rpmstore/config/etc` and they are replayed during boot on `/etc`.
- Any script created in the `/etc` directory persists across reloads. For example, a third-party service script created under `/etc/init.d/` brings up the apps during reload.



**Note** The rules in iptables are not persistent across reboots when they are modified in a bash-shell.

To make the modified iptables persistent, see [Making an Iptable Persistent Across Reloads, on page 166](#).

## Installing RPM from VSH

### Package Addition

NX-OS feature RPMs can also be installed by using the VSH CLIs.

#### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>show install packages</b>	Displays the packages and versions that already exist.
<b>Step 2</b>	<b>install add ?</b>	Determine supported URIs.
<b>Step 3</b>	<b>install add rpm-packagename</b>	The <b>install add</b> command copies the package file to a local storage device or network server.

#### Example

The following example shows how to activate the Chef RPM:

```
switch# show install packages
switch# install add ?
WORD          Package name
bootflash:   Enter package uri
ftp:         Enter package uri
http:        Enter package uri
modflash:    Enter package uri
```

```

scp:      Enter package uri
sftp:     Enter package uri
tftp:     Enter package uri
usb1:     Enter package uri
usb2:     Enter package uri
volatile: Enter package uri
switch# install add
bootflash:chef-12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.e15.x86_64.rpm
[#####] 100%
Install operation 314 completed successfully at Thu Aug 6 12:58:22 2015

```

### What to do next

When you are ready to activate the package, go to [Package Activation](#).



**Note** Adding and activating an RPM package can be accomplished in a single command:

```

switch#
install add bootflash:chef-12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.e15.x86_64.rpm
activate

```

## Package Activation

### Before you begin

The RPM has to have been previously added.

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<b>show install inactive</b>	Displays the list of packages that were added and not activated.
<b>Step 2</b>	<b>install activate</b> <i>rpm-packagename</i>	Activates the package.

### Example

The following example shows how to activate a package:

```

switch# show install inactive
Boot image:
    NXOS Image: bootflash:///yumcli6.bin

Inactive Packages:
    sysinfo-1.0.0-7.0.3.x86_64
Loaded plugins: downloadonly, importpubkey, localrpmDB, patchaction, patching,
                : protect-packages
Available Packages
chef.x86_64      12.0.0alpha.2+20150319234423.git.1608.b6eb10f-1.e15  thirdparty
eigrp.lib32_n3600 1.0.0-r0                                           groups-rep
o

```

```

sysinfo.x86_64      1.0.0-7.0.3
switch# install activate chef-12.0-1.e15.x86_64.rpm
[#####] 100%
Install operation completed successfully at Thu Aug  6 12:46:53 2015

```

patching

## Deactivating Packages

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<code>install deactivate <i>package-name</i></code>	Deactivates the RPM package.

### Example

The following example shows how to deactivate the Chef RPM package:

```
switch# install deactivate chef
```

## Removing Packages

### Before you begin

Deactivate the package before removing it. Only deactivated RPM packages can be removed.

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<code>install remove <i>package-name</i></code>	Removes the RPM package.

### Example

The following example shows how to remove the Chef RPM package:

```
switch# install remove chef-12.0-1.e15.x86_64.rpm
```

## Displaying Installed Packages

### Procedure

	Command or Action	Purpose
<b>Step 1</b>	<code>show install packages</code>	Displays a list of the installed packages.

**Example**

The following example shows how to display a list of the installed packages:

```
switch# show install packages
```

## Displaying Detail Logs

**Procedure**

	Command or Action	Purpose
<b>Step 1</b>	show tech-support install	Displays the detail logs.

**Example**

The following example shows how to display the detail logs:

```
switch# show tech-support install
```

## Upgrading a Package

**Procedure**

	Command or Action	Purpose
<b>Step 1</b>	install add <i>package-name</i> activate upgrade	Upgrade a package.

**Example**

The following example show how to upgrade a package:

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate ?
downgrade Downgrade package
forced Non-interactive
upgrade Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate upgrade
[#####] 100%
Install operation completed successfully at Thu Aug 6 12:46:53 2015
```

## Downgrading a Package

**Procedure**

	Command or Action	Purpose
<b>Step 1</b>	install add <i>package-name</i> activate downgrade	Downgrade a package.

### Example

The following example shows how to downgrade a package:

```
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate ?
downgrade Downgrade package
forced      Non-interactive
upgrade     Upgrade package
switch# install add bootflash:bgp-1.0.1-r0.lib32_n3600.rpm activate downgrade
[#####] 100%
Install operation completed successfully at Thu Aug 6 12:46:53 2015
```

## Third-Party Applications

### NX-OS

For more information about NX-API REST API object model specifications, see <https://developer.cisco.com/media/dme/index.html>

### collectd

collectd is a daemon that periodically collects system performance statistics and provides multiple means to store the values, such as RRD files. Those statistics can then be used to find current performance bottlenecks (i.e. performance analysis) and predict future system load (that is, capacity planning).

For additional information, see <https://collectd.org>.

### Ganglia

Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on thousands of clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

For additional information, see <http://ganglia.info>.

### Iperf

Iperf was developed by NLANR/DAST to measure maximum TCP and UDP bandwidth performance. Iperf allows the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss.

For additional information, see <http://sourceforge.net/projects/iperf/> or <http://iperf.sourceforge.net>.

## LLDP

The link layer discover protocol (LLDP) is an industry standard protocol designed to supplant proprietary link layer protocols such as EDP or CDP. The goal of LLDP is to provide an inter-vendor compatible mechanism to deliver link layer notifications to adjacent network devices.

For more information, see <https://vincentbernat.github.io/lldpd/index.html>.

## Nagios

Nagios is open source software that monitors network services (through ICMP, SNMP, SSH, FTP, HTTP etc), host resources (CPU load, disk usage, system logs, etc.), and alert services for servers, switches, applications, and services through the Nagios remote plugin executor (NRPE) and through SSH or SSL tunnels.

For more information, see <https://www.nagios.org/>.

## OpenSSH

OpenSSH is an open-source version of the SSH connectivity tools that encrypts all traffic (including passwords) to effectively eliminate eavesdropping, connection hijacking, and other attacks. OpenSSH provides secure tunneling capabilities and several authentication methods, and supports all SSH protocol versions.

For more information, see <http://www.openssh.com>.

## Quagga

Quagga is a network routing software suite that implements various routing protocols. Quagga daemons are configured through a network accessible CLI called a "vty".



---

**Note** Only Quagga BGP has been validated.

---

For more information, see <http://www.nongnu.org/quagga/>.

## Splunk

Splunk is a web based data collection, analysis, and monitoring tool that has a search, visualization and pre-packaged content for use-cases. The raw data is sent to the Splunk server using the Splunk Universal Forwarder. Universal Forwarders provide reliable, secure data collection from remote sources and forward that data into the Splunk Enterprise for indexing and consolidation. They can scale to tens of thousands of remote systems, collecting terabytes of data with minimal impact on performance.

For additional information, see [http://www.splunk.com/en\\_us/download/universal-forwarder.html](http://www.splunk.com/en_us/download/universal-forwarder.html).

## tcollector

tcollector is a client-side process that gathers data from local collectors and pushes the data to Open Time Series Database (OpenTSDB).

tcollector has the following features:

- Runs data collectors and collates the data,
- Manages connections to the time series database (TSD),
- Eliminates the need to embed TSD code in collectors,
- De-duplicates repeated values, and
- Handles wire protocol work.

For additional information, see [http://opentsdb.net/docs/build/html/user\\_guide/utilities/tcollector.html](http://opentsdb.net/docs/build/html/user_guide/utilities/tcollector.html).

## tcpdump

Tcpdump is a CLI application that prints out a description of the contents of packets on a network interface that match the boolean expression; the description is preceded by a time stamp, printed, by default, as hours, minutes, seconds, and fractions of a second since midnight. It can also be run with the `-w` flag, which causes it to save the packet data to a file for later analysis, and/or with the `-r` flag, which causes it to read from a saved packet file rather than to read packets from a network interface. It can also be run with the `-V` flag, which causes it to read a list of saved packet files. In all cases, only packets that match expression will be processed by tcpdump.

For more information, see <http://www.tcpdump.org/manpages/tcpdump.1.html>.

## Tshark

TShark is a network protocol analyzer on the CLI. It lets you capture packet data from a live network, or read packets from a previously saved capture file. You can either print a decoded form of those packets to the standard output or write the packets to a file. TShark's native capture file format is the pcap format, which is also the format used by **tcpdump** and various other tools. Tshark can be used within the Guest Shell 2.1 after removing the `cap_net_admin` file capability.

```
setcap
cap_net_raw=ep /sbin/dumppcap
```



---

**Note** This command must be run within the Guest Shell.

---

For more information, see <https://www.wireshark.org/docs/man-pages/tshark.html>.







# CHAPTER 10

## Ansible

---

- [Prerequisites](#), on page 105
- [About Ansible](#), on page 105
- [Cisco Ansible Module](#), on page 105

### Prerequisites

Go to [https://docs.ansible.com/ansible/intro\\_installation.html](https://docs.ansible.com/ansible/intro_installation.html) for installation requirements for supported control environments.

### About Ansible

Ansible is an open-source IT automation engine that automates cloud provisioning, configuration management, application deployment, intraservice orchestration, and other IT needs.

Ansible uses small programs that are called Ansible modules to make API calls to your nodes, and apply configurations that are defined in playbooks.

By default, Ansible represents what machines it manages using a simple INI file that puts all your managed machines in groups of your own choosing.

More information can be found from Ansible:

Ansible	<a href="https://www.ansible.com/">https://www.ansible.com/</a>
Ansible Automation Solutions. Includes installation instructions, playbook instructions and examples, module lists, and so on.	<a href="https://docs.ansible.com/">https://docs.ansible.com/</a>

### Cisco Ansible Module

There are multiple Cisco NX-OS-supported modules and playbooks for Ansible, as per the following table of links:

NX-OS developer landing page.	<a href="#">Configuration Management Tools</a>
-------------------------------	--

Ansible NX-OS playbook examples	<a href="#">Repo for ansible nxos playbooks</a>
Ansible NX-OS network modules	<a href="#">nxos network modules</a>



# CHAPTER 11

## Puppet Agent

This chapter includes the following sections:

- [About Puppet, on page 107](#)
- [Prerequisites, on page 107](#)
- [Puppet Agent NX-OS Environment, on page 108](#)
- [ciscopuppet Module, on page 108](#)

### About Puppet

The Puppet software package, developed by Puppet Labs, is an open source automation toolset for managing servers and other resources. The Puppet software accomplishes server and resource management by enforcing device states, such as configuration settings.

Puppet components include a puppet agent which runs on the managed device (node) and a Puppet Master (server). The Puppet Master typically runs on a separate dedicated server and serves multiple devices. The operation of the puppet agent involves periodically connecting to the Puppet Master, which in turn compiles and sends a configuration manifest to the agent. The agent reconciles this manifest with the current state of the node and updates state that is based on differences.

A puppet manifest is a collection of property definitions for setting the state on the device. The details for checking and setting these property states are abstracted so that a manifest can be used for more than one operating system or platform. Manifests are commonly used for defining configuration settings, but they also can be used to install software packages, copy files, and start services.

More information can be found from Puppet Labs:

Puppet Labs	<a href="https://puppetlabs.com">https://puppetlabs.com</a>
Puppet Labs FAQ	<a href="https://puppet.com/products/faq">https://puppet.com/products/faq</a>
Puppet Labs Documentation	<a href="https://puppet.com/docs">https://puppet.com/docs</a>

### Prerequisites

The following are prerequisites for the Puppet Agent:

- You must have a Cisco device and operating system software release that supports the installation.

- Cisco Nexus 3500 Series switch
  - Cisco Nexus 3100 Series switch.
  - Cisco Nexus 3000 Series switch.
  - Cisco NX-OS release 7.0(3)I2(1) or later.
- You must have the required disk storage available on the device for virtual services installation and deployment of Puppet Agent.
    - A minimum of 450MB free disk space on bootflash.
  - You must have Puppet Master server with Puppet 4.0 or later.
  - You must have Puppet Agent 4.0 or later.

## Puppet Agent NX-OS Environment

The Puppet Agent software must be installed on a Cisco Nexus platform in the Guest Shell (the Linux container environment running CentOS). The Guest Shell provides a secure, open execution environment that is decoupled from the host.

Starting with the Cisco NX-OS Release 9.2(1), the Bash-shell (native WindRiver Linux environment underlying NX-OS) install of Puppet Agent is no longer supported.

The following provides information about agent-software download, installation, and setup:

Puppet Agent: Installation & Setup on Cisco Nexus switches (Manual Setup)	<a href="https://github.com/cisco/cisco-network-puppet-module/blob/develop/docs/README-agent-install.md">https://github.com/cisco/cisco-network-puppet-module/blob/develop/docs/README-agent-install.md</a>
---	---

## ciscopuppet Module

The ciscopuppet module is a Cisco developed open-source software module. It interfaces between the abstract resources configuration in a puppet manifest and the specific implementation details of the Cisco Nexus NX-OS operating system and platform. This module is installed on the Puppet Master and is required for puppet agent operation on Cisco Nexus switches.

The ciscopuppet module is available on Puppet Forge.

The following provide additional information about the ciscopuppet module installation procedures:

ciscopuppet Module location (Puppet Forge)	<a href="https://forge.puppetlabs.com/puppetlabs/ciscopuppet">https://forge.puppetlabs.com/puppetlabs/ciscopuppet</a>
Resource Type Catalog	<a href="https://github.com/cisco/cisco-network-puppet-module/tree/master#resource-by-tech">https://github.com/cisco/cisco-network-puppet-module/tree/master#resource-by-tech</a>
ciscopuppet Module: Source Code Repository	<a href="https://github.com/cisco/cisco-network-puppet-module/tree/master">https://github.com/cisco/cisco-network-puppet-module/tree/master</a>

ciscopuppet Module: Setup & Usage	<a href="#">Cisco Puppet Module::README.md</a>
Puppet Labs: Installing Modules	<a href="https://docs.puppetlabs.com/puppet/latest/reference/modules_installing.html">https://docs.puppetlabs.com/puppet/latest/reference/modules_installing.html</a>
Puppet NX-OS Manifest Examples	<a href="https://github.com/cisco/cisco-network-puppet-module/tree/master/examples">https://github.com/cisco/cisco-network-puppet-module/tree/master/examples</a>
NX-OS developer landing page.	<a href="#">Configuration Management Tools</a>





## CHAPTER 12

# Using Chef Client with Cisco NX-OS

This chapter includes the following sections:

- [About Chef, on page 111](#)
- [Prerequisites, on page 111](#)
- [Chef Client NX-OS Environment, on page 112](#)
- [cisco-cookbook, on page 112](#)

## About Chef

Chef is an open-source software package that is developed by Chef Software, Inc. The software package is a systems and cloud infrastructure automation framework that deploys servers and applications to any physical, virtual, or cloud location, no matter the size of the infrastructure. Each organization consists of one or more workstations, a single server, and every node that the chef-client has configured and is maintaining. Cookbooks and recipes are used to tell the chef-client how each node should be configured. The chef-client, which is installed on every node, does the actual configuration.

A Chef cookbook is the fundamental unit of configuration and policy distribution. A cookbook defines a scenario and contains everything that is required to support that scenario, including libraries, recipes, files, and more. A Chef recipe is a collection of property definitions for setting state on the device. The details for checking and setting these property states are abstracted away so that a recipe may be used for more than one operating system or platform. While recipes are commonly used for defining configuration settings, they also can be used to install software packages, copy files, start services, and more.

The following references provide more information from Chef:

Topic	Link
Chef home	<a href="https://www.chef.io">https://www.chef.io</a>
Chef overview	<a href="https://docs.chef.io/chef_overview.html">https://docs.chef.io/chef_overview.html</a>
Chef documentation (all)	<a href="https://docs.chef.io/">https://docs.chef.io/</a>

## Prerequisites

The following are prerequisites for Chef:

- You must have a Cisco device and operating system software release that supports the installation:
  - Cisco Nexus 3500 Series switch
  - Cisco Nexus 3100 Series switch
  - Cisco Nexus 3000 Series switch
  - Cisco NX-OS Release 7.0(3)I2(1) or higher
- You must have the required disk storage available on the device for Chef deployment:
  - A minimum of 500 MB free disk space on bootflash
- You need a Chef server with Chef 12.4.1 or higher.
- You need Chef Client 12.4.1 or higher.

## Chef Client NX-OS Environment

The chef-client software must be installed on a Cisco Nexus platform in the Guest Shell (the Linux container environment running CentOS). This software provides a secure, open execution environment that is decoupled from the host.

Starting with the Cisco NX-OS Release 9.2(1), the Bash-shell (native WindRiver Linux environment underlying NX-OS) install of chef-client is no longer supported.

The following documents provide step-by-step guidance about agent-software download, installation, and setup:

Topic	Link
Chef Client: Installation and setup on Cisco Nexus platform (manual setup)	<a href="#">cisco-cookbook::README-install-agent.md</a>
Chef Client: Installation and setup on Cisco Nexus platform (automated installation using the Chef provisioner)	<a href="#">cisco-cookbook::README-chef-provisioning.md</a>

## cisco-cookbook

cisco-cookbook is a Cisco-developed open-source interface between the abstract resources configuration in a Chef recipe and the specific implementation details of the Cisco NX-OS and platforms. This cookbook is installed on the Chef Server and is required for proper Chef Client operation on Cisco Nexus devices.

The cisco-cookbook can be found on Chef Supermarket.

The following documents provide more detail for cisco-cookbook and generic cookbook installation procedures:

Topic	Link
cisco-cookbook location	<a href="https://supermarket.chef.io/cookbooks/cisco-cookbook">https://supermarket.chef.io/cookbooks/cisco-cookbook</a>



Topic	Link
Resource Type Catalog	<a href="https://github.com/cisco/cisco-network-chef-cookbook/tree/master#resource-by-tech">https://github.com/cisco/cisco-network-chef-cookbook/tree/master#resource-by-tech</a>
cisco-cookbook: Source Code Repository	<a href="https://github.com/cisco/cisco-network-chef-cookbook/tree/master">https://github.com/cisco/cisco-network-chef-cookbook/tree/master</a>
cisco-cookbook: Setup and usage	<a href="https://github.com/cisco/cisco-network-chef-cookbook/blob/master/README.md#setup">https://github.com/cisco/cisco-network-chef-cookbook/blob/master/README.md#setup</a>
Chef Supermarket	<a href="https://supermarket.chef.io">https://supermarket.chef.io</a>
Chef NX-OS Manifest Examples	<a href="https://github.com/cisco/cisco-network-chef-cookbook/tree/master/recipes">https://github.com/cisco/cisco-network-chef-cookbook/tree/master/recipes</a>





## CHAPTER 13

# Nexus Application Development - ISO

---

This chapter contains the following sections:

- [About ISO, on page 115](#)
- [Installing the ISO, on page 115](#)
- [Using the ISO to Build Applications, on page 116](#)
- [Using RPM to Package an Application, on page 117](#)

## About ISO

The ISO image is a bootable Wind River 5 environment that includes the necessary tools, libraries, and headers to build and RPM-package third-party applications to run natively on a Cisco Nexus switch.

The content is not exhaustive, and it might be required that the user download and build any dependencies needed for any particular application.



---

**Note** Some applications are ready to be downloaded and used from the Cisco devhub website and do not require building.

---

## Installing the ISO

The ISO image is available for download at: [http://devhub.cisco.com/artifactory/simple/open-nxos/7.0-3-12-1/x86\\_64/satori-vm-intel-xeon-core.iso](http://devhub.cisco.com/artifactory/simple/open-nxos/7.0-3-12-1/x86_64/satori-vm-intel-xeon-core.iso).

The ISO is intended to be installed as a virtual machine. Use instructions from your virtualization vendor to install the ISO.

### Procedure

---

**Step 1** (Optional) VMware-based installation.

The ISO image installation on a VMWare virtual machine requires the virtual disk to be configured as SATA and not SCSI.

**Step 2** (Optional) QEMU-based installation.

Enter the following commands:

```
bash$ qemu-img create satori.img 10G
bash$ qemu-system-x86_64 -cdrom ./satori-vm-intel-xeon-core.iso -hda ./satori.img -m 8192
```

Once the ISO starts to boot, a menu is displayed. Choose the **Graphics Console Install** option. This installs to the virtual HD. Once the install is complete, the virtual machine must be rebooted.

**What to do next**

To login to the system, enter **root** as the login and **root** as the password.

Using the ISO to Build Applications Most of the build procedures that work with the SDK, and Linux in general, also apply to the ISO environment. However, there is no shell environment script to run. The default paths should be fine to use the tools installed. The source code for applications needs to be obtained through the usual mechanisms such as a source tar file or git repository.

Build the source code:

```
bash$ tar --xvzf example-lib.tgz
bash$ mkdir example-lib-install
bash$ cd example-lib/
bash$ ./configure --prefix=/path/to/example_lib_install
bash$ make

bash$ make install
```

## Using the ISO to Build Applications

Most of the build procedures that work with the SDK, and Linux in general, also apply to the ISO environment. However, there is no shell environment script to run. The default paths should be fine to use the tools installed. The source code for applications needs to be obtained through the usual mechanisms such as a source tar file or git repository.

**Procedure**

Build the source code.

- a) **tar -xvzf example-lib.tgz**
- b) **mkdir example-lib-install**
- c) **cd example-lib/**
- d) **./configure --prefix=path\_to\_example-lib-install**
- e) **make**
- f) **make install**

The steps are normal Linux.

**Example:**

The following example shows how to build the source code:

```
bash$ tar -xvzf example-lib.tgz
bash$ mkdir example-lib-install
bash$ cd example-lib/
bash$ ./configure --prefix=<path_to_example-lib-install>
bash$ make
bash$ make install
```

---

## Using RPM to Package an Application

If the application successfully builds using "make", then it can be packaged into an RPM.



---

### Note RPM and spec files

The RPM package format is designed to package up all files (binaries, libraries, configurations, documents, etc) that are needed for a complete install of the given application. The process of creating an RPM file is therefore somewhat non-trivial. To aid in the RPM build process, a .spec file is used that controls everything about the build process.



---

**Note** Many third-party applications are available on the internet in the form of source code packaged into tarballs. In many cases, these tarballs will include a .spec file to help with RPM build process. Unfortunately, many of these .spec files are not updated as frequently as the source code itself. Even worse, sometimes there is no spec file at all. In these cases the spec file may need editing or even creating from scratch so that RPMs can be built.

---





## CHAPTER 14

# Nexus Application Development - SDK

---

This chapter contains the following sections:

- [About the Cisco SDK, on page 119](#)
- [Installing the SDK, on page 119](#)
- [Procedure for Installation and Environment Initialization, on page 120](#)
- [Using the SDK to Build Applications, on page 121](#)
- [Using RPM to Package an Application, on page 122](#)
- [Creating an RPM Build Environment, on page 123](#)
- [Using General RPM Build Procedure, on page 123](#)
- [Example to Build RPM for collectd with No Optional Plug-Ins, on page 124](#)
- [Example to Build RPM for collectd with Optional Curl Plug-In, on page 125](#)

## About the Cisco SDK

The Cisco SDK is a development kit based on Yocto 1.2. It contains all of the tools needed to build applications for execution on a Cisco Nexus switch running the NX-OS Release 7.0(3)I2(1). The basic components are the C cross-compiler, linker, libraries, and header files that are commonly used in many applications. The list is not exhaustive, and it might be required that you download and build any dependencies needed for any particular application. Note that some applications are ready to be downloaded and used from the Cisco devhub website and do not require building. The SDK can be used to build RPM packages which may be directly installed on a switch.

## Installing the SDK

The following lists the system requirements:

- The SDK can run on most modern 64-bit x86\_64 Linux systems. It has been verified on CentOS 7 and Ubuntu 14.04. Install and run the SDK under the Bash shell.
- The SDK includes binaries for both 32-bit and 64-bit architectures, so it must be run on an x86\_64 Linux system that also has 32-bit libraries installed.

## Procedure

---

Check if the 32-bit libraries are installed:

### Example:

```
bash$ ls /lib/ld-linux.so.2
```

If this file exists, then 32-bit libraries should be installed already. Otherwise, install 32-bit libraries as follows:

- For CentOS 7:

```
bash$ sudo yum install glibc.i686
```

- For Ubuntu 14.04:

```
bash$ sudo apt-get install gcc-multilib
```

---

# Procedure for Installation and Environment Initialization

The SDK is available for download at: [http://devhub.cisco.com/artifactory/simple/open-nxos/7.0-3-I2-1/x86\\_64/wrlinux-5.0.1.13-eglibc-x86\\_64-n9000-nxos-image-rpm-sdk-sdk.sh](http://devhub.cisco.com/artifactory/simple/open-nxos/7.0-3-I2-1/x86_64/wrlinux-5.0.1.13-eglibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh)

This file is a self-extracting archive that installs the SDK into a directory of your choice. You are prompted for a path to an SDK installation directory.

```
bash$ ./wrlinux-5.0.1.13-eglibc-x86_64-n9000-nxos-image-rpm-sdk-sdk.sh
Enter target directory for SDK (default: /opt/windriver/wrlinux/5.0-n9000):
/path/to/sdk_install_directory
You are about to install the SDK to "/path/to/sdk_install_directory". ProceedY/n?Y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
bash$
```

Use the **source environment-setup-x86\_64-wrs-linux** command to add the SDK-specific paths to your shell environment. This must be done for each shell you intend to use with the SDK. This is the key to setting up the SDK in order to use the correct versions of the build tools and libraries.

## Procedure

---

**Step 1** Browse to the installation directory.

**Step 2** Enter the following command at the Bash prompt:

```
bash$ source environment-setup-x86_64-wrs-linux
```

---



# Using the SDK to Build Applications

Many of the common Linux build processes work for this scenario. Use the techniques that are best suited for your situation.

The source code for an application package can be retrieved in various ways. For example, you can get the source code either in tar file form or by downloading from a git repository where the package resides.

The following are examples of some of the most common cases.

**(Optional) Verify that the application package builds using standard configure/make/make install.**

```
bash$ tar --xvzf example-app.tgz
bash$ mkdir example-lib-install
bash$ cd example-app/
bash$ ./configure --prefix=/path/to/example-app-install
bash$ make
bash$ make install
```

Sometimes it is necessary to pass extra options to the `./configure` script, for example to specify which optional components and dependencies are needed. Passing extra options depends entirely on the application being built.

## Example - Build Ganglia and its dependencies

In this example, we build ganglia, along with the third-party libraries that it requires - libexpat, libapr, and libconfuse.

### libexpat

```
bash$ wget 'http://downloads.sourceforge.net/project/expat/expat/2.1.0/expat-2.1.0.tar.gz'
bash$ mkdir expat-install
bash$ tar xvzf expat-2.1.0.tar.gz
bash$ cd expat-2.1.0
bash$ ./configure --prefix=/home/sdk-user/expat-install
bash$ make
bash$ make install
bash$ cd ..
```

### libapr

```
bash$ wget 'http://www.eu.apache.org/dist/apr/apr-1.5.2.tar.gz'
bash$ mkdir apr-install
bash$ tar xvzf apr-1.5.2.tar.gz
bash$ cd apr-1.5.2
bash$ ./configure --prefix=/home/sdk-user/apr-install
bash$ make
bash$ make install
bash$ cd ..
```

### libconfuse



**Note** confuse requires the extra `--enable-shared` option to `./configure`, otherwise it builds a statically linked library instead of the required shared library.

```

bash$ wget 'http://savannah.nongnu.org/download/confuse/confuse-2.7.tar.gz'
bash$ mkdir confuse-install
bash$ tar xvzf confuse-2.7.tar.gz
bash$ cd confuse-2.7
bash$ ./configure --prefix=/home/sdk-user/confuse-install --enable-shared
bash$ make
bash$ make install
bash$ cd ..

```

### ganglia




---

**Note** The locations to all the required libraries are passed to `./configure`.

---

```

bash$ wget
'http://downloads.sourceforge.net/project/ganglia/ganglia%20monitoring%20core/3.7.2/ganglia-3.7.2.tar.gz'
bash$ mkdir ganglia-install
bash$ tar xvzf ganglia-3.7.2.tar.gz
bash$ cd ganglia-3.7.2
bash$ ./configure --with-libexpat=/home/sdk-user/expat-install
--with-libapr=/home/sdk-user/apr-install/bin/apr-1-config
--with-libconfuse=/home/sdk-user/confuse-install --prefix=/home/sdk-user/ganglia-install
bash$ make
bash$ make install
bash$ cd ..

```

## Using RPM to Package an Application

If the application successfully builds using "make", then it can be packaged into an RPM.




---

**Note** **RPM and spec files**

The RPM package format is designed to package up all files (binaries, libraries, configurations, documents, etc) that are needed for a complete install of the given application. The process of creating an RPM file is therefore somewhat non-trivial. To aid in the RPM build process, a `.spec` file is used that controls everything about the build process.

---




---

**Note** Many third-party applications are available on the internet in the form of source code packaged into tarballs. In many cases, these tarballs will include a `.spec` file to help with RPM build process. Unfortunately, many of these `.spec` files are not updated as frequently as the source code itself. Even worse, sometimes there is no spec file at all. In these cases the spec file may need editing or even creating from scratch so that RPMs can be built.

---

# Creating an RPM Build Environment

Before using the SDK to build RPMs, an RPM build directory structure must be created, and some RPM macros set.

## Procedure

**Step 1** Create the directory structure:

```
bash$ mkdir rpmbuild
bash$ cd rpmbuild
bash$ mkdir BUILD RPMS SOURCES SPECS SRPMS
```

**Step 2** Set the topdir macro to point to the directory structure created above:

```
bash$ echo "_topdir ${PWD}" > ~/.rpmmacros
```

**Note** This step assumes that the current user does not already have a `.rpmmacros` file that is already set up. If it is inconvenient to alter an existing `.rpmmacros` file, then the following may be added to all `rpmbuild` command lines:

```
--define "_topdir ${PWD}"
```

**Step 3** Refresh the RPM DB:

```
bash$ rm /path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm/__.db.*
bash$ rpm --rebuilddb
```

**Note** The `rpm` and `rpmbuild` tools in the SDK have been modified to use `/path/to/sdk/sysroots/x86_64-wrlinuxsdk-linux/var/lib/rpm` as the RPM database instead of the normal `/var/lib/rpm`. This modification prevents any conflicts with the RPM database for the host when not using the SDK and removes the need for root access. After SDK installation, the SDK RPM database must be rebuilt through this procedure.

# Using General RPM Build Procedure

General RPM Build procedure is as follows:

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES http://<URL of example-app tarball>
bash$ # determine location of spec file in tarball:
bash$ tar tf SOURCES/example-app.tar.bz2 | grep '.spec$'
bash$ tar xkvf SOURCES/example-app.tar.bz2 example-app/example-app.spec
bash$ mv example-app/example-app.spec SPECS/
bash$ rm -rf example-app
bash$ rpmbuild -v --bb SPECS/example-app.spec
```

The result is a binary RPM in `RPMS/` that can be copied to the switch and installed. Installation and configuration of applications can vary. Refer to the application documents for those instructions.

This rpmbuild and installation on the switch is required for every software package that is required to support the application. If a software dependency is required that is not already included in the SDK, the source code must be obtained and the dependencies built. On the build machine, the package can be built manually for verification of dependencies. The following example is the most common procedure:

```
bash$ tar xkzf example-lib.tgz
bash$ mkdir example-lib-install
bash$ cd example-lib/
bash$ ./configure --prefix=/path/to/example-lib-install
bash$ make
bash$ make install
```

These commands place the build files (binaries, headers, libraries, and so on) into the installation directory. From here, you can use standard compiler and linker flags to pick up the location to these new dependencies. Any runtime code, such as libraries, are required to be installed on the switch also, so packaging required runtime code into an RPM is required.




---

**Note** There are many support libraries already in RPM form on the Cisco devhub website.

---

## Example to Build RPM for collectd with No Optional Plug-Ins

Download source tarball and extract spec file:

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```

There are four spec files in this tarball. The Red Hat spec file is the most comprehensive and is the only one that contains the correct collectd version. We will use it as an example.

This spec file sets the RPM up to use /sbin/chkconfig to install collectd. However on a Nexus switch, you will use the /usr/sbin/chkconfig instead. Edit the following edited in the spec file:

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

*collectd* has numerous optional plug-ins. This spec file enables many plug-ins by default. Many plug-ins have external dependencies, so options to disable these plug-ins must be passed to the **rpmbuild** command line. Instead of typing out one long command line, we can manage the options in a Bash array as follows:

```
bash$ rpmbuild_opts=()
bash$ for rmddep in \
> amqp apache ascent bind curl curl_xml dbi ipmi java memcached mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>   rpmbuild_opts+=("--without")
>   rpmbuild_opts+=(${rmddep})
```

```
> done
bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

It is then passed to rpmbuild as follows to start the entire build and RPM package process:

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

You can then find the resulting RPMs for collectd in the RPMS directory.

These RPM files can now be copied to the switch and installed from the switch Bash shell:

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
```

## Example to Build RPM for collectd with Optional Curl Plug-In

The collectd curl plug-in has libcurl as a dependency.

In order to satisfy this link dependency during the RPM build process, it is necessary to download and build curl under the SDK:

```
bash$ wget --no-check-certificate http://curl.haxx.se/download/curl-7.24.0.tar.gz
bash$ tar xkvf curl-7.24.0.tar.gz
bash$ cd curl-7.24.0
bash$ ./configure --without-ssl --prefix /path/to/curl-install
bash$ make
bash$ make install
bash$ cd ..
```



**Note** The curl binaries and libraries are installed to `/path/to/curl-install`. This directory will be created if it does not already exist, so you must have write permissions for the current user. Next, download the source tarball and extract the spec file. This step is exactly the same as in the collectd example for no plugins.

```
bash$ wget --no-check-certificate --directory-prefix=SOURCES
https://collectd.org/files/collectd-5.5.0.tar.bz2
bash$ tar tf SOURCES/collectd-5.5.0.tar.bz2 | grep '.spec$'
collectd-5.5.0/contrib/redhat/collectd.spec
collectd-5.5.0/contrib/aix/collectd.spec
collectd-5.5.0/contrib/sles10.1/collectd.spec
collectd-5.5.0/contrib/fedora/collectd.spec
bash$ tar xkvf SOURCES/collectd-5.5.0.tar.bz2 collectd-5.5.0/contrib/redhat/collectd.spec
bash$ mv collectd-5.5.0/contrib/redhat/collectd.spec SPECS/
bash$ rm -rf collectd-5.5.0
```



**Note** There are four spec files in this tarball. The Red Hat spec file is the most comprehensive, and it is the only one to contain the correct collectd version. We will use that one as an example.

This spec file sets the RPM up to use `/sbin/chkconfig` to install collectd. However on a Cisco Nexus switch, you must use `/usr/sbin/chkconfig` instead, so the following can be edited in the spec file:

```
bash$ sed -r -i.bak 's%(^|\s)/sbin/chkconfig%\1/usr/sbin/chkconfig%' SPECS/collectd.spec
```

Here a deviation from the previous example is encountered. The collectd rpmbuild process needs to know the location of libcurl. Edit the collectd spec file to add the following.

Find the string `%configure` in `SPECS/collectd.spec`. This line and those following it define the options that rpmbuild will pass to the `./configure` script.

Add the following option:

```
--with-libcurl=/path/to/curl-install/bin/curl-config \
```

Next a Bash array is built again to contain the rpmbuild command options. Note the following differences:

- `curl` is removed from the list of plug-ins not to be built
- The addition of `--with curl=force`

```
bash$ rpmbuild_opts=()
bash$ for rmdep in \
> amqp apache ascent bind curl_xml dbi ipmi java memcached mysql nginx \
> notify_desktop notify_email nut openldap perl pinba ping postgresql python \
> rrdtool sensors snmp varnish virt write_http write_riemann
> do
>   rpmbuild_opts+=("--without")
>   rpmbuild_opts+=(${rmdep})
> done
bash$ rpmbuild_opts+=("--with")
bash$ rpmbuild_opts+=("curl=force")bash$ rpmbuild_opts+=(--nodeps)
bash$ rpmbuild_opts+=(--define)
bash$ rpmbuild_opts+=("_unpackaged_files_terminate_build 0")
```

It is then passed to rpmbuild as follows to start the entire build and RPM package process:

```
bash$ rpmbuild "${rpmbuild_opts[@]}" -bb SPECS/collectd.spec
```

The resulting RPMs in the RPMs directory will now also include collectd-curl. These RPM files can now be copied to the switch and installed from the switch Bash shell:

```
bash$ rpm --noparentdirs -i /bootflash/collectd-5.5.0-1.ia32e.rpm
bash$ rpm --noparentdirs -i /bootflash/collectd-curl-5.5.0-1.ia32e.rpm
```



# CHAPTER 15

## NX-SDK

---

- [About the NX-SDK, on page 127](#)
- [Install the NX-SDK, on page 128](#)
- [Building and Packaging C++ Applications, on page 128](#)
- [Installing and Running Custom Applications, on page 131](#)

## About the NX-SDK

The Cisco NX-OS SDK (NX-SDK) is a C++ abstraction/plugin library layer that streamlines access to infrastructure for automation and custom native application creation, such as generating custom:

- CLIs.
- Syslogs.
- Event and Error managers.
- Inter-application communication.
- High availability (HA).
- Route manager.

The NX-SDK also supports Python bindings.



---

**Note** For Cisco Nexus NX-OS 7.0(3)I6(1) and earlier versions, the NX-SDK is not supported on Cisco Nexus 3000 switches.

---

### Requirements

The NX-SDK has the following requirements:

- Docker

# Install the NX-SDK

## Procedure

---

- Step 1** **Note** The Cisco SDK is required for applications started in VSH.  
The Cisco SDK is optional for applications started in Bash.

(Optional) Build the Cisco SDK RPM to persist on switch reloads and from standby mode.

- a) Pull the Docker image for Ubuntu 14.04+ or Centos 6.7+ from <https://hub.docker.com/r/dockercisco/nxsdk>.
- b) Source for a 32-bit environment:

**Example:**

```
export ENXOS_SDK_ROOT=/enxos-sdk
cd $ENXOS_SDK_Root
source environment-setup-x86-linux
```

- Step 2** Clone the NX-SDK toolkit from <https://github.com/CiscoDevNet/NX-SDK.git>.

**Example:**

```
git clone https://github.com/CiscoDevNet/NX-SDK.git
```

---

## What to do next

The following references to the API can be found in `$PWD/nxsdk` and includes the following:

- The NX-SDK public C++ classes and APIs,
- Example applications, and
- Example Python applications.

# Building and Packaging C++ Applications

The following instructions describes how to build and package your custom C++ NX-OS application.

## Procedure

---

- Step 1** Build your application files..
- a) Building a C++ application requires adding your source files to the `Makefile`

**Example:**

The example below uses the `customCliApp.cpp` file from `/examples`



```
...
##Directory Structure
...
EXNXSDK_BIN:= customCliApp
...
```

- b) Build the C++ application using the **make** command.

**Example:**

```
$PWD/nxsdk# make clean
$PWD/nxsdk# make all
```

**Step 2** (Optional) Package your application.

**Auto-generate RPM package**

Custom RPM packages for your applications are required to run on VSH and allow you to specify whether a given application persists on switch reloads or system switchovers. Use the following to create a custom specification file for your application.

**Note** RPM packaging is required to be done within the provided ENXOS Docker image.

- a) Use the [rpm\\_gen.py](#) script to auto-generate RPM package for a custom application.

**Example:**

Specify the **-h** option of the script to display the usages of the script.

```
/NX-SDK# python scripts/rpm_gen.py -h
```

- b) By default, NXSDK\_ROOT is set to /NX-SDK. If NX-SDK is installed in another location other than the default, then you must set NXSDK\_ROOT env to the appropriate location for the script to run correctly.

**Example:**

```
export NXSDK_ROOT=<absolute-path-to-NX-SDK>
```

Example of Auto-generate RPM package for C++ App examples/customCliApp.cpp

```
/NX-SDK/scripts# python rpm_gen.py CustomCliApp
#####

Generating rpm package...

Executing(%prep): /bin/sh -e /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ exit 0
Executing(%build): /bin/sh -e
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ exit 0
Executing(%install): /bin/sh -e
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ /bin/rm -rf
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../var/tmp/customCliApp-root
+ /bin/mkdir -p
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../var/tmp/customCliApp-root//isan/bin
```

```

+ cp -R /NX-SDK/bin /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/..
/../../var/tmp/customCliApp-root//isan/bin
+ exit 0
Processing files: customCliApp-1.0-7.03.I6.1.x86_64
Requires: libc.so.6 libc.so.6(GLIBC 2.0) 3.0) Libc.so.6(GLIBC_2.1.3) libdl.so.2 libgcc_s.so.1
libgcc_s.so.1(GCC_3.0) libm.so.6 libnxsdk.so libstdc++.so.6 libstdc++.so.6 (CXXAB1 1.3)
libstdc++.so.6(GLIBCXX 3.4) libstdc++.so.6(GLIBCXX_3.4.14) rtld(GNU HASH)
Checking for unpackaged file(s):
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/check-files
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../var/tmp/customCliApp-root
Wrote:
/enxos-sdk/sysroots/X86_64-wrlinuxsdk-linux/usr/src/rpm/SRPMs/customCliApp-1.0-7.0.3.I6.1.src-rpm

Wrote:
/enxos-sdk/sysroots/X86_64-wrlinuxsdk-linux/usr/src/rpm/RPMS/x86_64/customCliApp-1.0-7.0.3.I6.1.x86_64.rpm
Executing($clean): /bin/sh -e
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/var/tmp/rpm-tmp.49266
+ umask 022
+ cd /enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../src/rpm/BUILD
+ /bin/rm -rf
/enxos-sdk/sysroots/x86_64-wrlinuxsdk-linux/usr/lib/rpm/../../var/tmp/customCliApp-root

RPM package has been built
#####

SPEC file: /NX-SDK/rpm/SPECS/customCliApp.spec
RPM file : /NX-SDK/rpm/RPMS/customCliApp-1.0-7.0.3.I6.1.x86_64.rpm

```

### Manually-generate RPM Package

Custom RPM packages for your applications are required to run on VSH and allow you to specify whether a given application persists on switch reloads or system switchovers. Use the following steps to create a custom specification file (\* .spec) for your application.

- a) Export the Cisco SDK RPM source to \$RPM\_ROOT.

**Example:**

```
export RPM_ROOT=$ENXOS_SDK_ROOT/sysroots/x86_64-wrlinuxsdk-linux/usr/src/rpm
```

- b) Enter the \$RPM\_ROOT directory.

**Example:**

```
ls $RPM_ROOT (BUILD RPMS SOURCES SPECS SRPMS)
```

- c) Create/edit your application-specific \* .spec file.

Refer to the customCliApp.spec file in the /rpm/SPECS directory for an example specification file.

**Note** We recommend installing application files to /isan/bin/nxsdk on the switch as per the example customCliApp.spec file.

**Example:**

```
vi $RPM_ROOT/SPECS/<application>.spec
```

- d) Build your RPM package.

**Example:**

```
rpm -ba $RPM_ROOT/SPECS/<application>.spec
```

A successful build will generate an RPM file in `$RPM_ROOT/RPMS/x86_64/`

## Installing and Running Custom Applications

You can install applications by copying binaries to the switch, or installing unpacking the binaries from the RPM package.



**Note** Only custom applications that are installed from RPM packages can persist on switch reload or system switchovers. We recommend reserving copying binaries to the switch for simple testing purposes.

To run NX-SDK apps inside the switch (on box), you must have the Cisco SDK build environment that is installed.



**Note** The Cisco SDK is required to start applications in VSH: VSH requires that all applications be installed through RPMs, which requires that being built in the Cisco SDK.

The Cisco SDK is not required for Python application.

The Cisco SDK is not required for C++ application, but is still recommended: Using `g++` to build applications and then copying the built files to the switch may pose stability risks as `g++` is not supported.

To install or run custom applications on the switch, use this procedure:

### Before you begin

The switch must have the NX-SDK enabled before running any custom application. Run **feature nxsdk** on the switch.

### Procedure

**Step 1** Install your application using either VSH or Bash.

To install your application using VSH, perform the following:

- a) Add the RPM package to the installer.

**Example:**

```
switch(config)# install add bootflash:<app-rpm-package>.rpm
```

- b) After installation, check if the RPM is listed as inactive.

**Example:**

```
switch(config)# show install inactive
```

- c) Activate the RPM package.

**Example:**

```
switch(config)# install activate <app-rpm-package>
```

d) After activation, check if the RPM is listed as active.

**Example:**

```
switch(config)# show install active
```

To install your application using Bash, run the following commands:

```
switch(config)# run bash sudo su
bash# yum install /bootflash/<app-rpm-package>.rpm
```

## Step 2

Start your application.

C++ applications can run from VSH or Bash.

- To run a C++ application in VSH, run the **nxsdk** command:

```
switch(config)# nxsdk service-name /<install directory>/<application>
```

**Note** If the application is installed in `/isan/bin/nxsdk`, the full file path is not required. You can use the **nxsdk service-name app-name** form of the command.

- To run a C++ application in Bash, start Bash then start the application.

```
switch(config)# run bash sudo su
bash# <app-full-path> &
```

Python applications can run from VSH or Bash.

- To run a Python application from VSH, run the **nxsdk** command:

```
switch(config)# nxsdk service-name <app-full-path>
```

**Note** The Python application must be made executable to start from VSH:

- Run **chmod +x app-full-path**
- Add `#!/isan/bin/nxpython` to the first link of your Python application.

- To run a Python application from Bash,

```
switch(config)# run bash sudo su
bash# /isan/bin/nxsdk <app-full-path>
```

**Note** By default, NX-SDK uses `/isan/bin/nxsdk` to run Python applications in Bash, but you can specify a different install directory if needed.

## Step 3

Run **show nxsdk internal service** to verify that your application is running

**Example:**

```
switch(config)# show nxsdk internal service
```

```
switch(config)# show nxsdk internal service
```

```
NXSDK total services (Max Allowed) : 2 (32)
NXSDK Default App Path             : /isan/bin/nxsdk
NXSDK Supported Versions           : 1.0
```

Service-name	Base App	Started (PID)	Version	RPM Package
-----	-----	-----	-----	-----

```

/isan/bin/capp1          nxsdk_app2      VSH (25270)      1.0
capp1-1.0-7.0.3.I6.1.x86_64
/isan/bin/TestApp.py    nxsdk_app3      BASH (27823)    -      -

```

**Step 4** Stop your application.

You can stop your application in the following ways:

- To stop all NX-SDK applications, run **no feature nxsdk**.
- To stop a specific application in VSH, run **no nxsdk service-name /install directory/application**
- To stop a specific application in Bash, run **application stop-event-loop**

**Step 5** Uninstall your application.

To uninstall the RPM from the switch using VSH, perform the following:

- a) Deactivate the active RPM package.

**Example:**

```
switch# install deactivate <app-rpm-package>
```

- b) Verify that the package is deactivated.

**Example:**

```
switch# show install inactive
```

- c) Remove the RPM package.

**Example:**

```
switch# install remove <app-rpm-package>
```

To uninstall the RPM from the switch using Bash, run **yum remove app-full-path**

---





# CHAPTER 16

## Using Docker with Cisco NX-OS

---

This chapter contains the following topics:

- [About Docker with Cisco NX-OS, on page 135](#)
- [Guidelines and Limitations, on page 135](#)
- [Prerequisites for Setting Up Docker Containers Within Cisco NX-OS, on page 136](#)
- [Starting the Docker Daemon, on page 136](#)
- [Configure Docker to Start Automatically, on page 137](#)
- [Starting Docker Containers: Host Networking Model, on page 138](#)
- [Starting Docker Containers: Bridged Networking Model, on page 139](#)
- [Mounting the bootflash and volatile Partitions in the Docker Container, on page 140](#)
- [Enabling Docker Daemon Persistence on Enhanced ISSU Switchover, on page 140](#)
- [Resizing the Docker Storage Backend, on page 141](#)
- [Stopping the Docker Daemon, on page 143](#)
- [Docker Container Security, on page 144](#)
- [Docker Troubleshooting, on page 145](#)

### About Docker with Cisco NX-OS

Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries. See <https://docs.docker.com/> for more information on Docker.

Beginning with Release 9.2(1), support is now added for using Docker within Cisco NX-OS on a switch.

The version of Docker that is included on the switch is 1.13.1. The Docker daemon is not running by default. You must start it manually or set it up to automatically restart when the switch boots up.

This section describes how to enable and use Docker in the specific context of the Cisco Nexus switch environment. Refer to the Docker documentation at <https://docs.docker.com/> for details on general Docker usage and functionality.

### Guidelines and Limitations

Following are the guidelines and limitations for using Docker on Cisco NX-OS on a switch:

- Docker functionality is supported on the Cisco Nexus 3000 series switches with at least 8 GB of system RAM.

# Prerequisites for Setting Up Docker Containers Within Cisco NX-OS

Following are the prerequisites for using Docker on Cisco NX-OS on a switch:

- Enable the host Bash shell. To use Docker on Cisco NX-OS on a switch, you must be the root user on the host Bash shell:

```
switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# feature bash-shell
```

- 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 in `/etc/sysconfig/docker`. For example:

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

- Verify that the switch clock is set correctly, or you might see the following error message:

```
x509: certificate has expired or is not yet valid
```

- Verify that the domain name and name servers are configured appropriately for the network and that it is reflected in the `/etc/resolv.conf` file:

```
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# vrf context management
switch(config-vrf)# ip domain-name ?
WORD Enter the default domain (Max Size 64)

switch(config-vrf)# ip name-server ?
A.B.C.D Enter an IPv4 address
A:B::C:D Enter an IPv6 address

root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
nameserver 171.70.168.183 #bleed
root@switch#
```

## Starting the Docker Daemon

When you start the Docker daemon for the first time, a fixed-size backend storage space is carved out in a file called `dockerpart` on the bootflash, which is then mounted to `/var/lib/docker`. If necessary, you can adjust the default size of this space by editing `/etc/sysconfig/docker` before you start the Docker daemon for the first time. You can also resize this storage space if necessary as described later on.

To start the Docker daemon:

### Procedure

- 
- Step 1** Load Bash and become superuser.



```
switch# run bash sudo su -
```

**Step 2** Start the Docker daemon.

```
root@switch# service docker start
```

**Step 3** Check the status.

```
root@switch# service docker status
dockerd (pid 3597) is running...
root@switch#
```

**Note** Once you start the Docker daemon, do not delete or tamper with the `dockerpart` file on the bootflash since it is critical to the docker functionality.

```
switch# dir bootflash:dockerpart
2000000000 Mar 14 12:50:14 2018 dockerpart
```

---

## Configure Docker to Start Automatically

You can configure the Docker daemon to always start up automatically when the switch boots up.

### Procedure

---

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Use the `chkconfig` utility to make the Docker service persistent.

```
root@switch# chkconfig --add docker
root@n9k-2#
```

**Step 3** Use the `chkconfig` utility to check the Docker service settings.

```
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
root@switch#
```

**Step 4** To remove the configuration so that Docker does not start up automatically:

```
root@switch# chkconfig --del docker
root@switch# chkconfig --list | grep docker
root@switch#
```

---

# Starting Docker Containers: Host Networking Model

If you want Docker containers to have access to all the host network interfaces, including data port and management, start the Docker containers with the `--network host` option. The user in the container can switch between the different network namespaces at `/var/run/netns` (corresponding to different VRFs configured in Cisco NX-OS) using the `ip netns exec <net_namespace> <cmd>`.

## Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Start the Docker container.

Following is an example of starting an Alpine Docker container on the switch and viewing all the network interfaces. The container is launched into the management network namespace by default.

```
root@switch# docker run --name=alpine --v /var/run/netns:/var/run/netns:ro,rslave --rm
--network host --cap-add SYS_ADMIN -it alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
management
default
/ #
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
3: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN group default
link/gre 0.0.0.0 brd 0.0.0.0
...
/ #
/ # ip netns exec default ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/16 scope host lo
valid_lft forever preferred_lft forever
2: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
link/ether 42:0d:9b:3c:d4:62 brd ff:ff:ff:ff:ff:ff
```

```
3: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default
link/ipip 0.0.0.0 brd 0.0.0.0
...
```

## Starting Docker Containers: Bridged Networking Model

If you want Docker containers to only have external network connectivity (typically through the management interface) and you don't necessarily care about visibility into a specific data port or other Cisco Nexus switch interface, you can start the Docker container with the default Docker bridged networking model. This is more secure than the host networking model described in the previous section since it also provides network namespace isolation.

### Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Start the Docker container.

Following is an example of starting an Alpine Docker container on the switch and installing the `iproute2` package.

```
root@switch# docker run -it --rm alpine
/ # apk --update add iproute2
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/x86_64/APKINDEX.tar.gz
(1/6) Installing libelf (0.8.13-r3)
(2/6) Installing libmnl (1.0.4-r0)
(3/6) Installing jansson (2.10-r0)
(4/6) Installing libnftnl-libs (1.0.8-r1)
(5/6) Installing iptables (1.6.1-r1)
(6/6) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
Executing busybox-1.27.2-r7.trigger
OK: 7 MiB in 17 packages
/ #
/ # ip netns list
/ #
```

**Step 3** Determine if you want to set up user namespace isolation.

For containers using the bridged networking model, you can also set up user namespace isolation to further improve security. See [Securing Docker Containers With User namespace Isolation, on page 144](#) for more information.

You can use standard Docker port options to expose a service from within the container, such as `sshd`. For example:

```
root@switch# docker run -d -p 18877:22 --name sshd_container sshd_ubuntu
```

This maps port 22 from within the container to port 18877 on the switch. The service can now be accessed externally through port 18877, as shown in the following example:

```
root@ubuntu-vm# ssh root@ip_address -p 18887
```

## Mounting the bootflash and volatile Partitions in the Docker Container

You can make the `bootflash` and `volatile` partitions visible in the Docker container by passing in the `-v /bootflash:/bootflash` and `-v /volatile:/volatile` options in the `run` command for the Docker container. This is useful if the application in the container needs access to files shared with the host, such as copying a new NX-OS system image to bootflash.



**Note** This `-v` command option allows for any directory to be mounted into the container and may result in information leaking or other accesses that may impact the operation of the NX-OS system. Limit this to resources such as `/bootflash` and `/volatile` that are already accessible using NX-OS CLI.

### Procedure

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Pass in the `-v /bootflash:/bootflash` and `-v /volatile:/volatile` options in the `run` command for the Docker container.

```
root@switch# docker run -v /bootflash:/bootflash -v /volatile:/volatile -it --rm alpine
/# ls /
bin          etc          media        root         srv          usr
bootflash   home         mnt          run          sys          var
dev          lib          proc         sbin         tmp          volatile
/ #
```

## Enabling Docker Daemon Persistence on Enhanced ISSU Switchover

You can have both the Docker daemon and any running containers persist on an Enhanced ISSU switchover. This is possible since the bootflash on which the backend Docker storage resides is the same and shared between both Active and Standby supervisors.

The Docker containers are disrupted (restarted) during the switchover, so they will not be running continuously.

### Procedure

---

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Before starting the switchover, use the `chkconfig` utility to make the Docker service persistent.

```
root@switch# chkconfig --add docker
root@n9k-2#
```

**Step 3** Start any containers using the `--restart unless-stopped` option so that they will be restarted automatically after the switchover.

The following example starts an Alpine container and configures it to always restart unless it is explicitly stopped or Docker is restarted:

```
root@switch# docker run -dit --restart unless-stopped alpine
root@n9k-2#
```

The Docker containers are disrupted (restarted) during the switchover, so they will not be running continuously.

---

## Resizing the Docker Storage Backend

After starting or using the Docker daemon, you can grow the size of the Docker backend storage space according to your needs.

### Procedure

---

**Step 1** Disable the Guest Shell.

If you do not disable the Guest Shell, it may interfere with the resize.

```
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
switch# 2018 Mar 15 17:16:55 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Deactivating
virtual service 'guestshell+'
2018 Mar 15 17:16:57 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully deactivated
virtual service 'guestshell+'
```

**Step 2** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 3** Get information on the current amount of storage space available.

```
root@switch# df -kh /var/lib/docker
Filesystem Size Used Avail Use% Mounted on
/dev/loop12 1.9G 7.6M 1.8G 1% /var/lib/docker
```

```
root@n9k-2#
```

**Step 4** Stop the Docker daemon.

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown
```

**Step 5** Get information on the current size of the Docker backend storage space (/bootflash/dockerpart).

```
root@switch# ls -l /bootflash/dockerpart
-rw-r--r-- 1 root root 2000000000 Mar 15 16:53 /bootflash/dockerpart
root@n9k-2#
```

**Step 6** Resize the Docker backend storage space.

For example, the following command increases the size by 500 megabytes:

```
root@switch# truncate -s +500MB /bootflash/dockerpart
root@n9k-2#
```

**Step 7** Get updated information on the size of the Docker backend storage space to verify that the resizing process was completed successfully.

For example, the following output confirms that the size of the Docker backend storage was successfully increased by 500 megabytes:

```
root@switch# ls -l /bootflash/dockerpart
-rw-r--r-- 1 root root 2500000000 Mar 15 16:54 /bootflash/dockerpart
root@n9k-2#
```

**Step 8** Check the size of the filesystem on /bootflash/dockerpart.

```
root@switch# e2fsck -f /bootflash/dockerpart
e2fsck 1.42.9 (28-Dec-2013)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/bootflash/dockerpart: 528/122160 files (0.6% non-contiguous), 17794/488281 blocks
```

**Step 9** Resize the filesystem on /bootflash/dockerpart.

```
root@switch# /sbin/resize2fs /bootflash/dockerpart
resize2fs 1.42.9 (28-Dec-2013)
Resizing the filesystem on /bootflash/dockerpart to 610351 (4k) blocks.
The filesystem on /bootflash/dockerpart is now 610351 blocks long.
```

**Step 10** Check the size of the filesystem on /bootflash/dockerpart again to confirm that the filesystem was successfully resized.

```
root@switch# e2fsck -f /bootflash/dockerpart
e2fsck 1.42.9 (28-Dec-2013)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/bootflash/dockerpart: 528/154736 files (0.6% non-contiguous), 19838/610351 blocks
```

**Step 11** Start the Docker daemon again.

```
root@switch# service docker start
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Starting dockerd with args '--debug=true':
```

**Step 12** Verify the new amount of storage space available.

```
root@switch# df -kh /var/lib/docker
Filesystem Size Used Avail Use% Mounted on
/dev/loop12 2.3G 7.6M 2.3G 1% /var/lib/docker
```

**Step 13** Exit out of Bash shell.

```
root@switch# exit
logout
switch#
```

**Step 14** Enable the Guest Shell, if necessary.

```
switch# guestshell enable

switch# 2018 Mar 15 17:12:53 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Activating virtual
service 'guestshell+'
switch# 2018 Mar 15 17:13:18 switch %$ VDC-1 %$ %VMAN-2-ACTIVATION_STATE: Successfully
activated virtual service 'guestshell+'
```

---

## Stopping the Docker Daemon

If you no longer wish to use Docker, follow the procedures in this topic to stop the Docker daemon.

### Procedure

---

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Stop the Docker daemon.

```
root@switch# service docker stop
Stopping dockerd: dockerd shutdown
```

**Step 3** Verify that the Docker daemon is stopped.

```
root@switch# service docker status
dockerd is stopped
root@switch#
```

**Note** You can also delete the `dockpart` file on the bootflash at this point, if necessary:

```
switch# delete bootflash:dockpart
Do you want to delete "/dockpart" ? (yes/no/abort) y
switch#
```

---

## Docker Container Security

Following are the Docker container security recommendations:

- Run in a separate user `namespace` if possible.
- Run in a separate network `namespace` if possible.
- Use `cgroups` to limit resources. An existing `cgroup` (`ext_ser`) is created to limit hosted applications to what the platform team has deemed reasonable for extra software running on the switch. Docker allows use of this and limiting per-container resources.
- Do not add unnecessary POSIX capabilities.

## Securing Docker Containers With User namespace Isolation

For containers using the bridged networking model, you can also set up user namespace isolation to further improve security. See <https://docs.docker.com/engine/security/usersns-remap/> for more information.

### Procedure

---

**Step 1** Determine if a `dockremap` group already exists on your system.

A `dockremap` user must already be set up on your system by default. If the `dockremap` group doesn't already exist, follow these steps to create it.

a) Enter the following command to create the `dockremap` group:

```
root@switch# groupadd dockremap -r
```

b) Create the `dockremap` user, unless it already exists:

```
root@switch# useradd dockremap -r -g dockremap
```

c) Verify that the `dockremap` group and the `dockremap` user were created successfully:

```
root@switch# id dockremap
uid=999(dockremap) gid=498(dockremap) groups=498(dockremap)
root@switch#
```

**Step 2** Add the desired re-mapped ID and range to the `/etc/subuid` and `/etc/subgid`.

For example:



```
root@switch# echo "dockremap:123000:65536" >> /etc/subuid
root@switch# echo "dockremap:123000:65536" >> /etc/subgid
```

**Step 3** Using a text editor, add the `--userns-remap=default` option to the `other_args` field in the `/etc/sysconfig/docker` file.

For example:

```
other_args="-debug=true --userns-remap=default"
```

**Step 4** Restart the Docker daemon, or start it if it is not already running, using `service docker [re]start`.

For example:

```
root@switch# service docker [re]start
```

Refer to the Docker documentation at <https://docs.docker.com/engine/security/userns-remap/> for more information on configuring and using containers with user namespace isolation.

---

## Moving the `cgroup` Partition

The `cgroup` partition for third-party services is `ext_ser`, which limits CPU usage to 25% per core. Cisco recommends that you run your Docker container under this `ext_ser` partition.

If the Docker container is run without the `--cgroup-parent=/ext_ser/` option, it can get up to the full 100% host CPU access, which can interfere with the regular operation of Cisco NX-OS.

### Procedure

---

**Step 1** Load Bash and become superuser.

```
switch# run bash sudo su -
```

**Step 2** Run the Docker container under the `ext_ser` partition.

For example:

```
root@switch# docker run --name=alpinerrun -v /var/run/netns:/var/run/netns:ro,rslave --rm
--network host --cgroup-parent=/ext_ser/ --cap-add SYS_ADMIN -it alpine
/ #
```

---

## Docker Troubleshooting

These topics describe issues that can arise with Docker containers and provides possible resolutions.

## Docker Fails to Start

**Problem:** Docker fails to start, showing an error message similar to the following:

```
switch# run bash
bash-4.3$ service docker start
Free bootflash: 39099 MB, total bootflash: 51771 MB
Carving docker bootflash storage: 2000 MB
2000+0 records in
2000+0 records out
2000000000 bytes (2.0 GB) copied, 22.3039 s, 89.7 MB/s
losetup: /dev/loop18: failed to set up loop device: Permission denied
mke2fs 1.42.9 (28-Dec-2013)
mkfs.ext4: Device size reported to be zero. Invalid partition specified, or
partition table wasn't reread after running fdisk, due to
a modified partition being busy and in use. You may need to reboot
to re-read your partition table.

Failed to create docker volume
```

**Possible Cause:** You might be running Bash as an admin user instead of as a root user.

**Solution:** Determine if you are running Bash as an admin user instead of as a root user:

```
bash-4.3$ whoami
admin
```

Exit out of Bash and run Bash as root user:

```
bash-4.3$ exit
switch# run bash sudo su -
```

## Docker Fails to Start Due to Insufficient Storage

**Problem:** Docker fails to start, showing an error message similar to the following, due to insufficient bootflash storage:

```
root@switch# service docker start
Free bootflash: 790 MB, total bootflash: 3471 MB
Need at least 2000 MB free bootflash space for docker storage
```

**Possible Cause:** You might not have enough free bootflash storage.

**Solution:** Free up space or adjust the `variable_dockerstrg` values in `/etc/sysconfig/docker` as needed, then restart the Docker daemon:

```
root@switch# cat /etc/sysconfig/docker
# Replace the below with your own docker storage backend boundary value (in MB)
# if desired.
boundary_dockerstrg=5000

# Replace the below with your own docker storage backend values (in MB) if
# desired. The smaller value applies to platforms with less than
# $boundary_dockerstrg total bootflash space, the larger value for more than
# $boundary_dockerstrg of total bootflash space.
small_dockerstrg=300
large_dockerstrg=2000
```

## Failure to Pull Images from Docker Hub (509 Certificate Expiration Error Message)

**Problem:** The system fails to pull images from the Docker hub with an error message similar to the following:

```
root@switch# docker pull alpine
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: x509: certificate has
expired or is not yet valid
```

**Possible Cause:** The system clock might not be set correctly.

**Solution:** Determine if the clock is set correctly or not:

```
root@n9k-2# sh clock
15:57:48.963 EST Thu Apr 25 2002
Time source is Hardware Calendar
```

Reset the clock, if necessary:

```
root@n9k-2# clock set hh:mm:ss { day month | month day } year
```

For example:

```
root@n9k-2# clock set 14:12:00 10 feb 2018
```

## Failure to Pull Images from Docker Hub (Client Timeout Error Message)

**Problem:** The system fails to pull images from the Docker hub with an error message similar to the following:

```
root@switch# docker pull alpine
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request canceled
while waiting for connection (Client.Timeout exceeded while awaiting headers)
```

**Possible Cause:** The proxies or DNS settings might not be set correctly.

**Solution:** Check the proxy settings and fix them, if necessary, then restart the Docker daemon:

```
root@switch# cat /etc/sysconfig/docker | grep proxy
#export http_proxy=http://proxy.esl.cisco.com:8080
#export https_proxy=http://proxy.esl.cisco.com:8080
root@switch# service docker [re]start
```

Check the DNS settings and fix them, if necessary, then restart the Docker daemon:

```
root@switch# cat /etc/resolv.conf
domain cisco.com #bleed
nameserver 171.70.168.183 #bleed
root@switch# # conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# vrf context management
switch(config-vrf)# ip domain-name ?
WORD Enter the default domain (Max Size 64)

switch(config-vrf)# ip name-server ?
A.B.C.D Enter an IPv4 address
```

```
A:::C:D Enter an IPv6 address
root@switch# service docker [re]start
```

## Docker Daemon or Containers Not Running On Switch Reload or Switchover

**Problem:** The Docker daemon or containers do not run after you have performed a switch reload or switchover.

**Possible Cause:** The Docker daemon might not be configured to persist on a switch reload or switchover.

**Solution:** Verify that the Docker daemon is configured to persist on a switch reload or switchover using the `chkconfig` command, then start the necessary Docker containers using the `--restart unless-stopped` option. For example, to start an Alpine container:

```
root@switch# chkconfig --add docker
root@switch#
root@switch# chkconfig --list | grep docker
docker 0:off 1:off 2:on 3:on 4:on 5:on 6:off
root@switch# docker run -dit --restart unless-stopped alpine
```

## Resizing of Docker Storage Backend Fails

**Problem:** An attempt to resize the Docker backend storage failed.

**Possible Cause:** You might not have Guest Shell disabled.

**Solution:** Use the following command to determine if Guest Shell is disabled:

```
root@switch# losetup -a | grep dockerpart
root@n9k-2#
```

The command should not display any output if Guest Shell is disabled.

Enter the following command to disable the Guest Shell, if necessary:

```
switch# guestshell disable
```

If you still cannot resize the Docker backend storage, you can delete `/bootflash/dockerpart`, then adjust the `[small_]large_dockerstrg` in `/etc/sysconfig/docker`, then start Docker again to get a fresh Docker partition with the size that you want.

## Docker Container Doesn't Receive Incoming Traffic On a Port

**Problem:** The Docker container doesn't receive incoming traffic on a port.

**Possible Cause:** The Docker container might be using a netstack port instead of a kstack port.

**Solution:** Verify that any ephemeral ports that are used by Docker containers are within the kstack range. Otherwise any incoming packets can get sent to netstack for servicing and dropped.

```
switch# show socket local-port-range
Kstack local port range (15001 - 58000)
Netstack local port range (58001 - 63535) and nat port range (63536 - 65535)
switch# conf t
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# sockets local-port-range <start_port> <end_port>
switch# run bash sudo su -
```

```
root@switch# cat /proc/sys/net/ipv4/ip_local_port_range
15001 58000
root@switch#
```

## Unable to See Data Port And/Or Management Interfaces in Docker Container

**Problem:** You are unable to see the data port or management interfaces in the Docker container.

**Solution:**

- Verify that the Docker container is started in the host network namespace with all host namespaces mapped in using the `-v /var/run/netns:/var/run/netns:ro,rslave --network host` options.
- Once in the container, you will be in the management network namespace by default. You can use the `ip netns` utility to move to the default (`init`) network namespace, which has the data port interfaces. The `ip netns` utility might need to be installed in the container using `yum`, `apk`, or something similar.

## General Troubleshooting Tips

**Problem:** You have other issues with Docker containers that were not resolved using other troubleshooting processes.

**Solution:**

- Look for `dockerd` debug output in `/var/log/docker` for any clues as to what is wrong.
- Verify that your switch has 8 GB or more of RAM. Docker functionality is not supported on any switch that has less than 8 GB of RAM.





## PART **III**

# NX-API

- [NX-API CLI, on page 153](#)
- [NX-API REST, on page 177](#)
- [NX-API Developer Sandbox, on page 179](#)







## CHAPTER 17

# NX-API CLI

---

- [About NX-API CLI, on page 153](#)
- [Using NX-API CLI, on page 154](#)
- [XML and JSON Supported Commands, on page 168](#)

## About NX-API CLI

On Cisco Nexus devices, command-line interfaces (CLIs) are run only on the device. NX-API CLI improves the accessibility of these CLIs by making them available outside of the switch by using HTTP/HTTPS. You can use this extension to the existing Cisco Nexus CLI system on the Cisco Nexus 3000 Series devices. NX-API CLI supports **show** commands, configurations, and Linux Bash.

NX-API CLI supports JSON-RPC.

The NX-API CLI also supports JSON/CLI Execution in Cisco Nexus 3500 Series devices.

## Transport

NX-API uses HTTP/HTTPS as its transport. CLIs are encoded into the HTTP/HTTPS POST body.

The NX-API backend uses the Nginx HTTP server. The Nginx process, and all of its children processes, are under Linux cgroup protection where the CPU and memory usage is capped. If the Nginx memory usage exceeds the cgroup limitations, the Nginx process is restarted and restored.



---

**Note** For the 7.x release, the Nginx process continues to run even after NX-API is disabled using the “no feature NXAPI” command. This is required for other management-related processes. In the 6.x release, all processes were killed when you ran the “no feature NXAPI” command, so this is a change in behavior in the 7.x release.

---

## Message Format



- 
- Note**
- NX-API XML output presents information in a user-friendly format.
  - NX-API XML does not map directly to the Cisco NX-OS NETCONF implementation.
  - NX-API XML output can be converted into JSON.
- 

## Security

NX-API supports HTTPS. All communication to the device is encrypted when you use HTTPS.

NX-API is integrated into the authentication system on the device. Users must have appropriate accounts to access the device through NX-API. NX-API uses HTTP basic authentication. All requests must contain the username and password in the HTTP header.



- 
- Note** You should consider using HTTPS to secure your user's login credentials.
- 

You can enable NX-API by using the **feature** manager CLI command. NX-API is disabled by default.

NX-API provides a session-based cookie, **nxapi\_auth** when users first successfully authenticate. With the session cookie, the username and password are included in all subsequent NX-API requests that are sent to the device. The username and password are used with the session cookie to bypass performing the full authentication process again. If the session cookie is not included with subsequent requests, another session cookie is required and is provided by the authentication process. Avoiding unnecessary use of the authentication process helps to reduce the workload on the device.



- 
- Note** A **nxapi\_auth** cookie expires in 600 seconds (10 minutes). This value is a fixed and cannot be adjusted.
- 



- 
- Note** NX-API performs authentication through a programmable authentication module (PAM) on the switch. Use cookies to reduce the number of PAM authentications, which reduces the load on the PAM.
- 

## Using NX-API CLI

The commands, command type, and output type for the Cisco Nexus 3000 Series devices are entered using NX-API by encoding the CLIs into the body of a HTTP/HTTPS POST. The response to the request is returned in XML or JSON output format.



- 
- Note** For more details about NX-API response codes, see [Table of NX-API Response Codes, on page 167](#).
-

You must enable NX-API with the **feature** manager CLI command on the device. By default, NX-API is disabled.

The following example shows how to configure and launch the NX-API CLI:

- Enable the management interface.

```
switch# conf t
switch(config)# interface mgmt 0
switch(config)# ip address 192.0.20.123/24
switch(config)# vrf context management
switch(config)# ip route 10.0.113.1/0 1.2.3.1
```

- Enable the NX-API **nxapi** feature.

```
switch# conf t
switch(config)# feature nxapi
```

The following example shows a request and its response in XML format:

Request:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ins_api>
  <version>0.1</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>session1</sid>
  <input>show switchname</input>
  <output_format>xml</output_format>
</ins_api>
```

Response:

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show switchname</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

The following example shows a request and its response in JSON format:

Request:

```
{
  "ins_api": {
    "version": "0.1",
    "type": "cli_show",
    "chunk": "0",
    "sid": "session1",
    "input": "show switchname",
    "output_format": "json"
  }
}
```

```
}

```

Response:

```
{
  "ins_api": {
    "type": "cli_show",
    "version": "0.1",
    "sid": "eoc",
    "outputs": {
      "output": {
        "body": {
          "hostname": "switch"
        },
        "input": "show switchname",
        "msg": "Success",
        "code": "200"
      }
    }
  }
}
```

## Escalate Privileges to Root on NX-API

For NX-API, the privileges of an admin user can escalate their privileges for root access.

The following are guidelines for escalating privileges:

- Only an admin user can escalate privileges to root.
- Escalation to root is password protected.

The following examples show how an admin escalates privileges to root and how to verify the escalation. Note that after becoming root, the **whoami** command shows you as admin; however, the admin account has all the root privileges.

First example:

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo su root ; whoami</input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>admin </body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
```

```
</ins_api>
```

Second example:

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo cat path_to_file </input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>[Contents of file]</body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

## NX-API Management Commands

You can enable and manage NX-API with the CLI commands listed in the following table.

**Table 4: NX-API Management Commands**

NX-API Management Command	Description
<b>feature nxapi</b>	Enables NX-API.
<b>no feature nxapi</b>	Disables NX-API.
<b>nxapi {http   https} port port</b>	Specifies a port.
<b>no nxapi {http   https}</b>	Disables HTTP/HTTPS.
<b>show nxapi</b>	Displays port and certificate information.

NX-API Management Command	Description
<b>nxapi certificate</b> { <b>https</b> cert certfile   <b>https</b> key keyfile} <i>filename</i>	Specifies the upload of the following: <ul style="list-style-type: none"> <li>• HTTPS certificate when <b>https</b>cert is specified.</li> <li>• HTTPS key when <b>https</b>key is specified.</li> </ul> Example of HTTPS certificate: <b>nxapi certificate https</b> cert certfile bootflash:cert.crt Example of HTTPS key: <b>nxapi certificate https</b> key keyfile bootflash:privkey.key
<b>nxapi certificate enable</b>	Enables a certificate.
<b>nxapi ssl-ciphers weak</b>	Starting with Cisco NX-OS Release 9.2(1), weak ciphers are disabled by default. Running this command changes the default behavior and enables the weak ciphers for NGINX. The <b>no</b> form of the command changes it to the default (by default, the weak ciphers are disabled).
<b>nxapi ssl-protocols</b> { <b>TLSv1.0</b> <b>TLSv1.1</b> <b>TLSv1.2</b> }	Starting with Cisco NX-OS Release 9.2(1), TLS1.0 is disabled by default. Running this command enables the TLS versions specified in the string, including the TLS1.0 that was disabled by default, if necessary. The <b>no</b> form of the command changes it to the default (by default, only TLS1.1 and TLS1.2 will be enabled).
<b>nxapi use-vrf</b> <i>vrf</i>	Specifies the default VRF, management VRF, or named VRF.
<b>ip netns exec management iptables</b>	Implements any access restrictions and can be run in management VRF. <p><b>Note</b> You must enable <b>feature bash-shell</b> and then run the command from Bash Shell. For more information on Bash Shell, see the chapter on Bash.</p> <p><i>Iptables is a command-line firewall utility that uses policy chains to allow or block traffic and almost always comes pre-installed on any Linux distribution.</i></p> <p><b>Note</b> For more information about making iptables persistent across reloads when they are modified in a bash-shell, see <a href="#">Making an Iptable Persistent Across Reloads, on page 166</a>.</p>

Following is an example of a successful upload of an HTTPS certificate:

```
switch(config)# nxapi certificate https certfile certificate.crt
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```



**Note** You must configure the certificate and key before enabling the certificate.

Following is an example of a successful upload of an HTTPS key:

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

## Working With Interactive Commands Using NX-API

To disable confirmation prompts on interactive commands and avoid timing out with an error code 500, prepend interactive commands with **terminal dont-ask**. Use `;` to separate multiple interactive commands, where each `;` is surrounded with single blank characters.

Following are several examples of interactive commands where **terminal dont-ask** is used to avoid timing out with an error code 500:

```
terminal dont-ask ; reload module 21
terminal dont-ask ; system mode maintenance
```

## NX-API Request Elements

NX-API request elements are sent to the device in XML format or JSON format. The HTTP header of the request must identify the content type of the request.

You use the NX-API elements that are listed in the following table to specify a CLI command:

**Table 5: NX-API Request Elements for XML or JSON Format**

NX-API Request Element	Description
version	Specifies the NX-API version.

NX-API Request Element	Description
<i>type</i>	<p>Specifies the type of command to be executed.</p> <p>The following types of commands are supported:</p> <ul style="list-style-type: none"> <li>• <b>cli_show</b> CLI <b>show</b> commands that expect structured output. If the command does not support XML output, an error message is returned.</li> <li>• <b>cli_show_array</b> CLI <b>show</b> commands that expect structured output. Only for show commands. Similar to <b>cli_show</b>, but with <b>cli_show_array</b>, data is returned as a list of one element, or an array, within square brackets [ ].</li> <li>• <b>cli_show_ascii</b> CLI <b>show</b> commands that expect ASCII output. This aligns with existing scripts that parse ASCII output. Users are able to use existing scripts with minimal changes.</li> <li>• <b>cli_conf</b> CLI configuration commands.</li> <li>• <b>bash</b> Bash commands. Most non-interactive Bash commands are supported by NX-API.</li> </ul> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• Each command is only executable with the current user's authority.</li> <li>• The pipe operation is supported in the output when the message type is ASCII. If the output is in XML format, the pipe operation is not supported.</li> <li>• A maximum of 10 consecutive <b>show</b> commands are supported. If the number of <b>show</b> commands exceeds 10, the 11th and subsequent commands are ignored.</li> <li>• No interactive commands are supported.</li> </ul>



NX-API Request Element	Description				
<i>chunk</i>	<p>Some <b>show</b> commands can return a large amount of output. For the NX-API client to start processing the output before the entire command completes, NX-API supports output chunking for <b>show</b> commands.</p> <p>Enable or disable chunk with the following settings:</p> <table border="1" data-bbox="824 485 1520 596"> <tr> <td data-bbox="824 485 938 539">0</td> <td data-bbox="938 485 1520 539">Do not chunk output.</td> </tr> <tr> <td data-bbox="824 539 938 596">1</td> <td data-bbox="938 539 1520 596">Chunk output.</td> </tr> </table> <p><b>Note</b></p> <ul style="list-style-type: none"> <li>• Only <b>show</b> commands support chunking. When a series of <b>show</b> commands are entered, only the first command is chunked and returned.</li> <li>• For the XML output message format (XML is the default.), special characters, such as &lt; or &gt;, are converted to form a valid XML message (&lt; is converted into &amp;lt; &gt; is converted into &amp;gt;).</li> </ul> <p>You can use XML SAX to parse the chunked output.</p> <p><b>Note</b> When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.</p>	0	Do not chunk output.	1	Chunk output.
0	Do not chunk output.				
1	Chunk output.				
<i>rollback</i>	<p>Valid only for configuration CLIs, not for show commands. Specifies the configuration rollback options. Specify one of the following options.</p> <ul style="list-style-type: none"> <li>• Stop-on-error—Stops at the first CLI that fails.</li> <li>• Continue-on-error—Ignores and continues with other CLIs.</li> <li>• Rollback-on-error—Performs a rollback to the previous state the system configuration was in.</li> </ul> <p><b>Note</b> The rollback element is available in the cli_conf mode when the input request format is XML or JSON.</p>				
<i>sid</i>	<p>The session ID element is valid only when the response message is chunked. To retrieve the next chunk of the message, you must specify a <i>sid</i> to match the <i>sid</i> of the previous response message.</p>				

NX-API Request Element	Description						
<i>input</i>	<p>Input can be one command or multiple commands. However, commands that belong to different message types should not be mixed. For example, <b>show</b> commands are cli_show message type and are not supported in cli_conf mode.</p> <p><b>Note</b> Except for <b>bash</b>, multiple commands are separated with ";". (The ; must be surrounded with single blank characters.)</p> <p>For <b>bash</b>, multiple commands are separated with ";". (The ; is <b>not</b> surrounded with single blank characters.)</p> <p>The following are examples of multiple commands:</p> <table border="1" data-bbox="786 688 1490 919"> <tbody> <tr> <td data-bbox="786 688 911 764">cli_show</td> <td data-bbox="911 688 1490 764">show version ; show interface brief ; show vlan</td> </tr> <tr> <td data-bbox="786 764 911 840">cli_conf</td> <td data-bbox="911 764 1490 840">interface Eth4/1 ; no shut ; switchport</td> </tr> <tr> <td data-bbox="786 840 911 919">bash</td> <td data-bbox="911 840 1490 919">cd /bootflash;mkdir new_dir</td> </tr> </tbody> </table>	cli_show	show version ; show interface brief ; show vlan	cli_conf	interface Eth4/1 ; no shut ; switchport	bash	cd /bootflash;mkdir new_dir
cli_show	show version ; show interface brief ; show vlan						
cli_conf	interface Eth4/1 ; no shut ; switchport						
bash	cd /bootflash;mkdir new_dir						
<i>output_format</i>	<p>The available output message formats are the following:</p> <table border="1" data-bbox="786 991 1490 1108"> <tbody> <tr> <td data-bbox="786 991 1024 1050">xml</td> <td data-bbox="1024 991 1490 1050">Specifies output in XML format.</td> </tr> <tr> <td data-bbox="786 1050 1024 1108">json</td> <td data-bbox="1024 1050 1490 1108">Specifies output in JSON format.</td> </tr> </tbody> </table>	xml	Specifies output in XML format.	json	Specifies output in JSON format.		
xml	Specifies output in XML format.						
json	Specifies output in JSON format.						



**Note** When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.

When JSON-RPC is the input request format, use the NX-API elements that are listed in the following table to specify a CLI command:

**Table 6: NX-API Request Elements for JSON-RPC Format**

NX-API Request Element	Description
<i>jsonrpc</i>	<p>A string specifying the version of the JSON-RPC protocol.</p> <p>Version must be 2.0.</p>

NX-API Request Element	Description
<i>method</i>	<p>A string containing the name of the method to be invoked.</p> <p>NX-API supports either:</p> <ul style="list-style-type: none"> <li>• <b>cli</b>—show or configuration commands</li> <li>• <b>cli_ascii</b>—show or configuration commands; output without formatting</li> <li>• <b>cli_array</b>—only for show commands; similar to <b>cli</b>, but with <b>cli_array</b>, data is returned as a list of one element, or an array, within square brackets, [ ].</li> </ul>
<i>params</i>	<p>A structured value that holds the parameter values used during the invocation of a method.</p> <p>It must contain the following:</p> <ul style="list-style-type: none"> <li>• <b>cmd</b>—CLI command</li> <li>• <b>version</b>—NX-API request version identifier</li> </ul>
<i>rollback</i>	<p>Valid only for configuration CLIs, not for show commands. Configuration rollback options. You can specify one of the following options.</p> <ul style="list-style-type: none"> <li>• Stop-on-error—Stops at the first CLI that fails.</li> <li>• Continue-on-error—Ignores the failed CLI and continues with other CLIs.</li> <li>• Rollback-on-error—Performs a rollback to the previous state the system configuration was in.</li> </ul>
<i>id</i>	<p>An optional identifier established by the client that must contain a string, number, or null value, if it is specified. The value should not be null and numbers contain no fractional parts. If a user does not specify the <i>id</i> parameter, the server assumes that the request is simply a notification, resulting in a no response, for example, <i>id</i> : 1</p>

## NX-API Response Elements

The NX-API elements that respond to a CLI command are listed in the following table:

**Table 7: NX-API Response Elements**

NX-API Response Element	Description
version	NX-API version.
type	Type of command to be executed.

NX-API Response Element	Description
sid	Session ID of the response. This element is valid only when the response message is chunked.
outputs	Tag that encloses all command outputs.  When multiple commands are in cli_show or cli_show_ascii, each command output is enclosed by a single output tag.  When the message type is cli_conf or bash, there is a single output tag for all the commands because cli_conf and bash commands require context.
output	Tag that encloses the output of a single command output.  For cli_conf and bash message types, this element contains the outputs of all the commands.
input	Tag that encloses a single command that was specified in the request. This element helps associate a request input element with the appropriate response output element.
body	Body of the command response.
code	Error code returned from the command execution.  NX-API uses standard HTTP error codes as described by the Hypertext Transfer Protocol (HTTP) Status Code Registry ( <a href="http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml">http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml</a> ).
msg	Error message associated with the returned error code.

## Restricting Access to NX-API

There are two methods for restricting HTTP and HTTPS access to a device: ACLs and iptables. The method that you use depends on whether you have configured a VRF for NX-API communication using the `nxapi use-vrf <vrf-name>` CLI command.

Use ACLs to restrict HTTP or HTTPS access to a device only if you have not configured NXAPI to use a specific VRF. For information about configuring ACLs, see the *Cisco Nexus 9000 Series NX-OS Security Configuration Guide*:

<https://www.cisco.com/c/en/us/support/switches/nexus-9000-series-switches/products-installation-and-configuration-guides-list.html>

If you have configured a VRF for NX-API communication, however, ACLs will not restrict HTTP or HTTPS access. Instead, create a rule for an iptable. For more information about creating a rule, see [Updating an iptable, on page 164](#).

## Updating an iptable

An iptable enables you to restrict HTTP or HTTPS access to a device when a VRF has been configured for NX-API communication. This section demonstrates how to add, verify, and remove rules for blocking HTTP and HTTPS access to an existing iptable.

## Procedure

### Step 1

To create a rule that blocks HTTP access:

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp --dport 80 -j DROP
```

### Step 2

To create a rule that blocks HTTPS access:

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp --dport 443 -j DROP
```

### Step 3

To verify the applied rules:

```
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source                destination           tcp dpt:http
DROP      tcp  --  anywhere              anywhere              tcp dpt:https
DROP      tcp  --  anywhere              anywhere              tcp dpt:https

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

### Step 4

To create and verify a rule that blocks all traffic with a 10.155.0.0/24 subnet to port 80:

```
bash-4.3# ip netns exec management iptables -A INPUT -s 10.155.0.0/24 -p tcp --dport 80 -j
DROP
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source                destination           tcp dpt:http
DROP      tcp  --  10.155.0.0/24        anywhere              tcp dpt:http

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

### Step 5

To remove and verify previously applied rules:

This example removes the first rule from INPUT.

```
bash-4.3# ip netns exec management iptables -D INPUT 1
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

**What to do next**

The rules in iptables are not persistent across reloads when they are modified in a bash-shell. To make the rules persistent, see [Making an Iptable Persistent Across Reloads, on page 166](#).

**Making an Iptable Persistent Across Reloads**

The rules in iptables are not persistent across reloads when they are modified in a bash-shell. This section explains how to make a modified iptable persistent across a reload.

**Before you begin**

You have modified an iptable.

**Procedure**

**Step 1** Create a file called iptables\_init.log in the /etc directory with full permissions:

```
bash-4.3# touch /etc/iptables_init.log; chmod 777 /etc/iptables_init.log
```

**Step 2** Create the /etc/sys/iptables file where your iptables changes will be saved:

```
bash-4.3# ip netns exec management iptables-save > /etc/sysconfig/iptables
```

**Step 3** Create a startup script called iptables\_init in the /etc/init.d directory with the following set of commands:

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:          iptables_init
# Required-Start:
# Required-Stop:
# Default-Start:    2 3 4 5
# Default-Stop:
# Short-Description: init for iptables
# Description:      sets config for iptables
#
#                   during boot time
### END INIT INFO

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
start_script() {
    ip netns exec management iptables-restore < /etc/sysconfig/iptables
    ip netns exec management iptables
    echo "iptables init script executed" > /etc/iptables_init.log
}
case "$1" in
    start)
        start_script
```

```

;;
stop)
;;
restart)
sleep 1
$0 start
;;
*)
echo "Usage: $0 {start|stop|status|restart}"
exit 1
esac
exit 0

```

**Step 4** Set the appropriate permissions to the startup script:

```
bash-4.3# chmod 777 /etc/init.d/iptables_init
```

**Step 5** Set the iptables\_init startup script to on with the chkconfig utility:

```
bash-4.3# chkconfig iptables_init on
```

The iptables\_init startup script will now execute each time that you perform a reload, making the iptable rules persistent.

## Table of NX-API Response Codes

The following are the possible NX-API errors, error codes, and messages of an NX-API response.



**Note** The standard HTTP error codes are at the Hypertext Transfer Protocol (HTTP) Status Code Registry (<http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml>).

**Table 8: NX-API Response Codes**

NX-API Response	Code	Message
SUCCESS	200	Success.
CUST_OUTPUT_PIPED	204	Output is piped elsewhere due to request.
BASH_CMD_ERR	400	Input Bash command error.
CHUNK_ALLOW_ONE_CMD_ERR	400	Chunking only allowed to one command.
CLI_CLIENT_ERR	400	CLI execution error.
CLI_CMD_ERR	400	Input CLI command error.
IN_MSG_ERR	400	Request message is invalid.
NO_INPUT_CMD_ERR	400	No input command.
PERM_DENY_ERR	401	Permission denied.

CONF_NOT_ALLOW_SHOW_ERR	405	Configuration mode does not allow <b>show</b> .
SHOW_NOT_ALLOW_CONF_ERR	405	Show mode does not allow configuration.
EXCEED_MAX_SHOW_ERR	413	Maximum number of consecutive show commands exceeded. The maximum is 10.
MSG_SIZE_LARGE_ERR	413	Response size too large.
BACKEND_ERR	500	Backend processing error.
CREATE_CHECKPOINT_ERR	500	Error creating a checkpoint.
FILE_OPER_ERR	500	System internal file operation error.
LIBXML_NS_ERR	500	System internal LIBXML NS error.
LIBXML_PARSE_ERR	500	System internal LIBXML parse error.
LIBXML_PATH_CTX_ERR	500	System internal LIBXML path context error.
MEM_ALLOC_ERR	500	System internal memory allocation error.
SERVER_BUSY_ERR	500	Request is rejected because the server is busy.
USER_NOT_FOUND_ERR	500	User not found from input or cache.
XML_TO_JSON_CONVERT_ERR	500	XML to JSON conversion error.
BASH_CMD_NOT_SUPPORTED_ERR	501	Bash command not supported.
CHUNK_ALLOW_XML_ONLY_ERR	501	Chunking allows only XML output.
JSON_NOT_SUPPORTED_ERR	501	JSON not supported due to large amount of output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
PIPE_OUTPUT_NOT_SUPPORTED_ERR	501	Pipe operation not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML is not allowed in input.
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	Response has large amount of output. JSON not supported.
STRUCT_NOT_SUPPORTED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Undefined.

## XML and JSON Supported Commands

The NX-OS supports redirecting the standard output of various **show** commands in the following structured output formats:



- XML
- JSON
- JSON Pretty, which makes the standard block of JSON-formatted output easier to read

Converting the standard NX-OS output to JSON, JSON Pretty, or XML format occurs on the NX-OS CLI by "piping" the output to a JSON or XML interpreter. For example, you can issue the **show ip access** command with the logical pipe (|) and specify JSON, JSON Pretty, or XML, and the NX-OS command output will be properly structured and encoded in that format. This feature enables programmatic parsing of the data and supports streaming data from the switch through software streaming telemetry. Most commands in Cisco NX-OS support JSON, JSON Pretty, and XML output.

Selected examples of this feature follow.

## About JSON (JavaScript Object Notation)

JSON is a light-weight text-based open standard designed for human-readable data and is an alternative to XML. JSON was originally designed from JavaScript, but it is language-independent data format. JSON Pretty format is also supported.

The two primary Data Structures that are supported in some way by nearly all modern programming languages are as follows:

- Ordered List :: Array
- Unordered List (Name/Value pair) :: Objects

JSON /XML output for a show command can also be accessed via sandbox.

### CLI Execution

```
BLR-VXLAN-NPT-CR-179# show cdp neighbors | json
{"TABLE_cdp_neighbor_brief_info": {"ROW_cdp_neighbor_brief_info": [{"ifindex": "83886080", "device_id": "SW-SPARSHA-SAVBU-F10", "intf_id": "mgmt0", "ttl": "148", "capability": ["switch", "IGMP_cnd_filtering"], "platform_id": "cisco WS-C2960 S-48TS-L", "port_id": "GigabitEthernet1/0/24"}, {"ifindex": "436207616", "device_id": "BLR-VXLAN-NPT-CR-178(FOC1745R01W)", "intf_id": "Ethernet1/1", "ttl": "166", "capability": ["router", "switch", "IGMP_cnd_filtering", "Supports-STP-Dispute"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}]}}
BLR-VXLAN-NPT-CR-179#
```

## Examples of XML and JSON Output

This example shows how to display the unicast and multicast routing entries in hardware tables in JSON format:

```
switch(config)# show hardware profile status | json
{"total_lpm": ["8191", "1024"], "total_host": "8192", "max_host4_limit": "4096", "max_host6_limit": "2048", "max_mcast_limit": "2048", "used_lpm_total": "9", "used_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_total": "0", "used_host_v4_lpm": "0", "used_host_v6_lpm": "0", "used_mcast": "0", "used_mcast_oif1": "2", "used_host_in_host_total": "13", "used_host4_in_host": "12", "used_host6_in_host": "1", "max_ecmp_table_limit": "64", "used_ecmp_table": "0", "mfib_fd_status": "Disabled", "mfib_fd_maxroute": "0", "mfib_fd_count": "0"}
switch(config)#
```

This example shows how to display the unicast and multicast routing entries in hardware tables in XML format:

```
switch(config)# show hardware profile status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://www.cisco.com/nxos:1.0:fib">
  <nf:data>
    <show>
      <hardware>
        <profile>
          <status>
            <__XML_OPT_Cmd_dynamic_tcama_status__>
              <__XML_OPT_Cmd_dynamic_tcama_status__readonly__>
                <__readonly__>
                  <total_lpm>8191</total_lpm>
                  <total_host>8192</total_host>
                  <total_lpm>1024</total_lpm>
                  <max_host4_limit>4096</max_host4_limit>
                  <max_host6_limit>2048</max_host6_limit>
                  <max_mcast_limit>2048</max_mcast_limit>
                  <used_lpm_total>9</used_lpm_total>
                  <used_v4_lpm>6</used_v4_lpm>
                  <used_v6_lpm>3</used_v6_lpm>
                  <used_v6_lpm_128>1</used_v6_lpm_128>
                  <used_host_lpm_total>0</used_host_lpm_total>
                  <used_host_v4_lpm>0</used_host_v4_lpm>
                  <used_host_v6_lpm>0</used_host_v6_lpm>
                  <used_mcast>0</used_mcast>
                  <used_mcast_oif1>2</used_mcast_oif1>
                  <used_host_in_host_total>13</used_host_in_host_total>
                  <used_host4_in_host>12</used_host4_in_host>
                  <used_host6_in_host>1</used_host6_in_host>
                  <max_ecmp_table_limit>64</max_ecmp_table_limit>
                  <used_ecmp_table>0</used_ecmp_table>
                  <mfib_fd_status>Disabled</mfib_fd_status>
                  <mfib_fd_maxroute>0</mfib_fd_maxroute>
                  <mfib_fd_count>0</mfib_fd_count>
                </__readonly__>
              </__XML_OPT_Cmd_dynamic_tcama_status__readonly__>
            </__XML_OPT_Cmd_dynamic_tcama_status__>
          </status>
        </profile>
      </hardware>
    </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

This example shows how to display LLDP timers configured on the switch in JSON format:

```
switch(config)# show lldp timers | json
{"ttl": "120", "reinit": "2", "tx_interval": "30", "tx_delay": "2", "hold_multiplier": "4", "notification_interval": "5"}
switch(config)#
```

This example shows how to display LLDP timers configured on the switch in XML format:

```

switch(config)# show lldp timers | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://www.cisco.com/nxos:1.0:lldp">
  <nf:data>
    <show>
      <lldp>
        <timers>
          <__XML__OPT_Cmd_lldp_show_timers__readonly__>
            <__readonly__>
              <ttl>120</ttl>
              <reinit>2</reinit>
              <tx_interval>30</tx_interval>
              <tx_delay>2</tx_delay>
              <hold_mplier>4</hold_mplier>
              <notification_interval>5</notification_interval>
            </__readonly__>
          </__XML__OPT_Cmd_lldp_show_timers__readonly__>
        </timers>
      </lldp>
    </show>
  </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#

```

This example shows how to display ACL statistics in XML format.

```

switch-1(config-acl)# show ip access-lists acl-test1 | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns="http://www.cisco.com/nxos:1.0:aclmgr" xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nf:data>
    <show>
      <__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
        <ip_ipv6_mac>ip</ip_ipv6_mac>
        <access-lists>
          <__XML__OPT_Cmd_show_acl_name>
            <name>acl-test1</name>
          <__XML__OPT_Cmd_show_acl_capture>
            <__XML__OPT_Cmd_show_acl_expanded>
              <__XML__OPT_Cmd_show_acl__readonly__>
                <__readonly__>
                  <TABLE_ip_ipv6_mac>
                    <ROW_ip_ipv6_mac>
                      <op_ip_ipv6_mac>ip</op_ip_ipv6_mac>
                      <show_summary>0</show_summary>
                      <acl_name>acl-test1</acl_name>
                      <statistics>enable</statistics>
                      <frag_opt_permit_deny>permit-all</frag_opt_permit_deny>
                    <TABLE_seqno>
                      <ROW_seqno>
                        <seqno>10</seqno>
                        <permitdeny>permit</permitdeny>
                        <ip>ip</ip>
                        <src_ip_prefix>192.0.2.1/24</src_ip_prefix>
                        <dest_any>any</dest_any>
                      </ROW_seqno>
                    </TABLE_seqno>
                  </ROW_ip_ipv6_mac>
                </TABLE_ip_ipv6_mac>
              </__readonly__>
            </__XML__OPT_Cmd_show_acl__readonly__>
          </access-lists>
        </__XML__OPT_Cmd_show_acl_ip_ipv6_mac>
      </show>
    </nf:data>
  </nf:rpc-reply>
]]>]]>

```

```

    </_XML_OPT_Cmd_show_acl_expanded>
  </_XML_OPT_Cmd_show_acl_capture>
</_XML_OPT_Cmd_show_acl_name>
</access-lists>
</_XML_OPT_Cmd_show_acl_ip_ipv6_mac>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch-1(config-acl)#

```

This example shows how to display ACL statistics in JSON format.

```

switch-1(config-acl)# show ip access-lists acl-test1 | json
{"TABLE_ip_ipv6_mac": {"ROW_ip_ipv6_mac": {"op_ip_ipv6_mac": "ip", "show_summary": "0", "acl_name": "acl-test1", "statistics": "enable", "frag_opt_permit_deny": "permit-all", "TABLE_seqno": {"ROW_seqno": {"seqno": "10", "permitdeny": "permit", "ip": "ip", "src_ip_prefix": "192.0.2.1/24", "dest_any": "any"}}}}}
switch-1(config-acl)#

```

The following example shows how to display the switch's redundancy status in JSON format.

```

switch-1# show system redundancy status | json
{"rdn_mode_admin": "HA", "rdn_mode_oper": "None", "this_sup": "(sup-1)", "this_sup_rdn_state": "Active, SC not present", "this_sup_sup_state": "Active", "this_sup_internal_state": "Active with no standby", "other_sup": "(sup-1)", "other_sup_rdn_state": "Not present"}
nxosv2#
switch-1#

```

The following example shows how to display the IP route summary in XML format.

```

switch-1# show ip route summary | xml
<?xml version="1.0" encoding="ISO-8859-1"?> <nf:rpc-reply
xmlns="http://www.cisco.com/nxos:1.0:urib" xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">

  <nf:data>
    <show>
      <ip>
        <route>
          <_XML_OPT_Cmd_urib_show_ip_route_command_ip>
            <_XML_OPT_Cmd_urib_show_ip_route_command_unicast>
              <_XML_OPT_Cmd_urib_show_ip_route_command_topology>
                <_XML_OPT_Cmd_urib_show_ip_route_command_l3vm-info>
                  <_XML_OPT_Cmd_urib_show_ip_route_command_rpf>
                    <_XML_OPT_Cmd_urib_show_ip_route_command_ip-addr>
                      <_XML_OPT_Cmd_urib_show_ip_route_command_protocol>
                        <_XML_OPT_Cmd_urib_show_ip_route_command_summary>
                          <_XML_OPT_Cmd_urib_show_ip_route_command_vrf>
                            <_XML_OPT_Cmd_urib_show_ip_route_command_readonly__>
                              <_readonly__>
                                <TABLE_vrf>
                                  <ROW_vrf>
                                    <vrf-name-out>default</vrf-name-out>
                                    <TABLE_addrf>
                                      <ROW_addrf>
                                        <addrf>ipv4</addrf>
                                        <TABLE_summary>
                                          <ROW_summary>
                                            <routes>938</routes>
                                            <paths>1453</paths>
                                            <TABLE_unicast>
                                              <ROW_unicast>
                                                <clientnameuni>am</clientnameuni>
                                                <best-paths>2</best-paths>
                                              </ROW_unicast>

```

```

        <ROW_unicast>
        <clientnameuni>local</clientnameuni>
        <best-paths>105</best-paths>
    </ROW_unicast>
    <ROW_unicast>
        <clientnameuni>direct</clientnameuni>
        <best-paths>105</best-paths>
    </ROW_unicast>
    <ROW_unicast>
        <clientnameuni>broadcast</clientnameuni>
        <best-paths>203</best-paths>
    </ROW_unicast>
    <ROW_unicast>
        <clientnameuni>ospf-10</clientnameuni>
        <best-paths>1038</best-paths>
    </ROW_unicast>
</TABLE_unicast>
<TABLE_route_count>
<ROW_route_count>
    <mask_len>8</mask_len>
    <count>1</count>
</ROW_route_count>
<ROW_route_count>
    <mask_len>24</mask_len>
    <count>600</count>
</ROW_route_count>
<ROW_route_count>
    <mask_len>31</mask_len>
    <count>13</count>
</ROW_route_count>
<ROW_route_count>
    <mask_len>32</mask_len>
    <count>324</count>
</ROW_route_count>
</TABLE_route_count>
</ROW_summary>
</TABLE_summary>
</ROW_addrf>
</TABLE_addrf>
</ROW_vrf>
</TABLE_vrf>
</__readonly__>
</__XML__OPT_Cmd_urib_show_ip_route_command__readonly__>
</__XML__OPT_Cmd_urib_show_ip_route_command_vrf>
</__XML__OPT_Cmd_urib_show_ip_route_command_summary>
</__XML__OPT_Cmd_urib_show_ip_route_command_protocol>
</__XML__OPT_Cmd_urib_show_ip_route_command_ip-addr>
</__XML__OPT_Cmd_urib_show_ip_route_command_rpf>
</__XML__OPT_Cmd_urib_show_ip_route_command_l3vm-info>
</__XML__OPT_Cmd_urib_show_ip_route_command_topology>
</__XML__OPT_Cmd_urib_show_ip_route_command_unicast>
</__XML__OPT_Cmd_urib_show_ip_route_command_ip>
</route>
</ip>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
switch-1#

```

The following example shows how to display the IP route summary in JSON format.

```

switch-1# show ip route summary | json
{"TABLE_vrf": {"ROW_vrf": {"vrf-name-out": "default", "TABLE_addrf": {"ROW_addrf": {"addrf": "ipv4", "TABLE_summary": {"ROW_summary": {"routes": "938", "paths": "

```

```
1453", "TABLE_unicast": {"ROW_unicast": [{"clientnameuni": "am", "best-paths": "2"},
{"clientnameuni": "local", "best-paths": "105"}, {"clientnameuni": "direct",
"best-paths": "105"}, {"clientnameuni": "broadcast", "best-paths": "203"}, {"clientnameuni":
"ospf-10", "best-paths": "1038"}]}, "TABLE_route_count": {"ROW_route_
count": [{"mask_len": "8", "count": "1"}, {"mask_len": "24", "count": "600"}, {"mask_len":
"31", "count": "13"}, {"mask_len": "32", "count": "324"}]}}}}}}}}
switch-1#
```

The following example shows how to display the IP route summary in JSON Pretty format.

```
switch-1# show ip route summary | json-pretty
{
  "TABLE_vrf": {
    "ROW_vrf": {
      "vrf-name-out": "default",
      "TABLE_addrf": {
        "ROW_addrf": {
          "addrf": "ipv4",
          "TABLE_summary": {
            "ROW_summary": {
              "routes": "938",
              "paths": "1453",
              "TABLE_unicast": {
                "ROW_unicast": [
                  {
                    "clientnameuni": "am",
                    "best-paths": "2"
                  },
                  {
                    "clientnameuni": "local",
                    "best-paths": "105"
                  },
                  {
                    "clientnameuni": "direct",
                    "best-paths": "105"
                  },
                  {
                    "clientnameuni": "broadcast",
                    "best-paths": "203"
                  },
                  {
                    "clientnameuni": "ospf-10",
                    "best-paths": "1038"
                  }
                ]
              },
            },
          },
        },
      },
    },
  },
  "TABLE_route_count": {
    "ROW_route_count": [
      {
        "mask_len": "8",
        "count": "1"
      },
      {
        "mask_len": "24",
        "count": "600"
      },
      {
        "mask_len": "31",
        "count": "13"
      },
      {
        "mask_len": "32",
        "count": "324"
      }
    ]
  }
}
```

```
}  
  }  
}  
switch-1#
```







## CHAPTER 18

# NX-API REST

---

- [About NX-API REST, on page 177](#)

## About NX-API REST

### NX-API REST

On Cisco Nexus devices, configuration is performed using command-line interfaces (CLIs) that run only on the device. NX-API REST improves the accessibility of the Nexus configuration by providing HTTP/HTTPS APIs that:

- Make specific CLIs available outside of the switch.
- Enable configurations that would require issuing many CLI commands by combining configuration actions in relatively few HTTP/HTTPS operations.

NX-API REST supports **show** commands, basic and advanced switch configurations, and Linux Bash.

NX-API REST uses HTTP/HTTPS as its transport. CLIs are encoded into the HTTP/HTTPS POST body. The NX-API REST backend uses the Nginx HTTP server. The Nginx process, and all of its children processes, are under Linux cgroup protection where the CPU and memory usage is capped. If the Nginx resource usage exceeds the cgroup limitations, the Nginx process is restarted and restored.

For more information about the NX-API REST SDK, see <https://developer.cisco.com/site/nx-api/documents/n3k-n9k-api-ref/>.





# CHAPTER 19

## NX-API Developer Sandbox

- NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2), on page 179
- NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later, on page 185

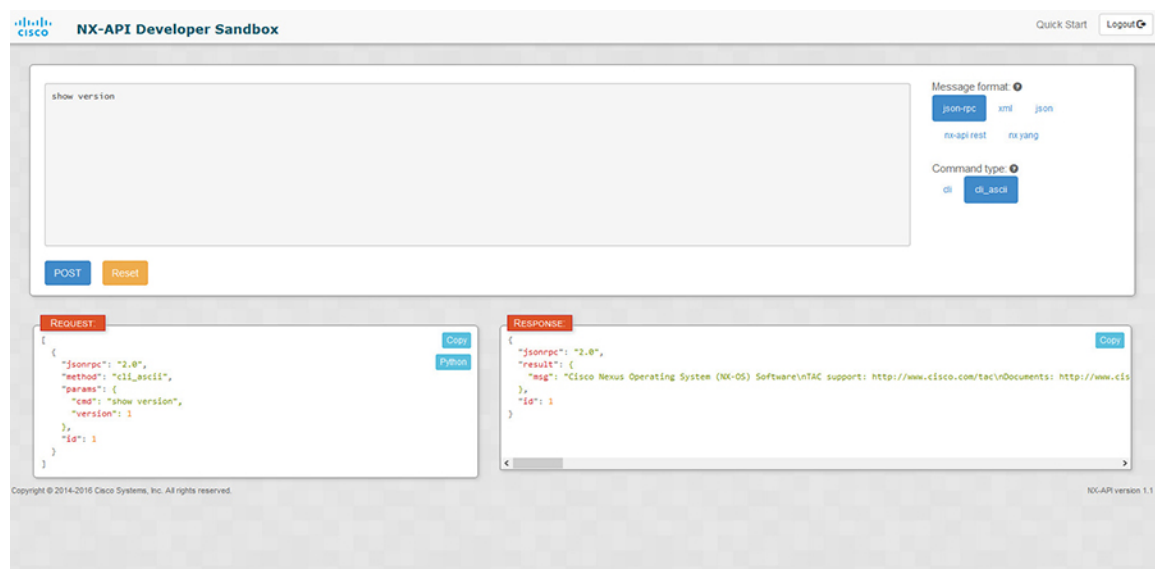
### NX-API Developer Sandbox: NX-OS Releases Prior to 9.2(2)

#### About the NX-API Developer Sandbox

The NX-API Developer Sandbox is a web form hosted on the switch. It translates NX-OS CLI commands into equivalent XML or JSON payloads, and converts NX-API REST payloads into their CLI equivalents.

The web form is a single screen with three panes — Command (top pane), Request, and Response — as shown in the figure.

**Figure 1: NX-API Developer Sandbox with Example Request and Output Response**



Controls in the Command pane allow you to choose a message format for a supported API, such as NX-API REST, and a command type, such as XML or JSON. The available command type options vary depending on the selected message format.

When you type or paste one or more CLI commands into the Command pane, the web form converts the commands into an API payload, checking for configuration errors, and displays the resulting payload in the Request pane. If you then choose to post the payload directly from the Sandbox to the switch, using the POST button in the Command pane, the Response pane displays the API response.

Conversely, when you type an NX-API REST designated name (DN) and payload into the Command pane and select the **nx-api rest** Message format and the **model** Command type, Developer Sandbox checks the payload for configuration errors, then the Response pane displays the equivalent CLIs.

## Guidelines and Limitations

Following are the guidelines and limitations for the Developer Sandbox:

- Clicking **POST** in the Sandbox commits the command to the switch, which can result in a configuration or state change.
- Some feature configuration commands are not available until their associated feature has been enabled.

## Configuring the Message Format and Command Type

The **Message Format** and **Command Type** are configured in the upper right corner of the Command pane (the top pane). For **Message Format**, choose the format of the API protocol that you want to use. The Developer Sandbox supports the following API protocols:

**Table 9: NX-OS API Protocols**

Protocol	Description
json-rpc	A standard lightweight remote procedure call (RPC) protocol that can be used to deliver NX-OS CLI commands in a JSON payload. The JSON-RPC 2.0 specification is outlined by <a href="http://jsonrpc.org">jsonrpc.org</a> .
xml	Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in an XML payload.
json	Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in a JSON payload.
nx-api rest	Cisco NX-API proprietary protocol for manipulating and reading managed objects (MOs) and their properties in the internal NX-OS data management engine (DME) model. For more information about the Cisco Nexus 3000 and 9000 Series NX-API REST SDK, see <a href="https://developer.cisco.com/site/cisco-nexus-nx-api-references/">https://developer.cisco.com/site/cisco-nexus-nx-api-references/</a> .
nx yang	The YANG ("Yet Another Next Generation") data modeling language for configuration and state data.

When the **Message Format** has been chosen, a set of **Command Type** options are presented just below the **Message Format** control. The **Command Type** setting can constrain the input CLI and can determine the **Request** and **Response** format. The options vary depending on the **Message Format** selection. For each **Message Format**, the following table describes the **Command Type** options:

Table 10: Command Types

Message format	Command type
json-rpc	<ul style="list-style-type: none"> <li>cli — show or configuration commands</li> <li>cli-ascii — show or configuration commands, output without formatting</li> </ul>
xml	<ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b> The bash shell must be enabled in the switch.</p>
json	<ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b> The bash shell must be enabled in the switch.</p>
nx-api rest	<ul style="list-style-type: none"> <li>cli — configuration commands</li> <li>model — DN and corresponding payload.</li> </ul>
nx yang	<ul style="list-style-type: none"> <li>json — JSON structure is used for payload</li> <li>xml — XML structure is used for payload</li> </ul>

### Output Chunking

In order to handle large show command output, some NX-API message formats support output chunking for show commands. In this case, an **Enable chunk mode** checkbox appears below the **Command Type** control along with a session ID (**SID**) type-in box.

When chunking is enabled, the response is sent in multiple "chunks," with the first chunk sent in the immediate command response. In order to retrieve the next chunk of the response message, you must send an NX-API request with **SID** set to the session ID of the previous response message.

## Using the Developer Sandbox

### Using the Developer Sandbox to Convert CLI Commands to Payloads



---

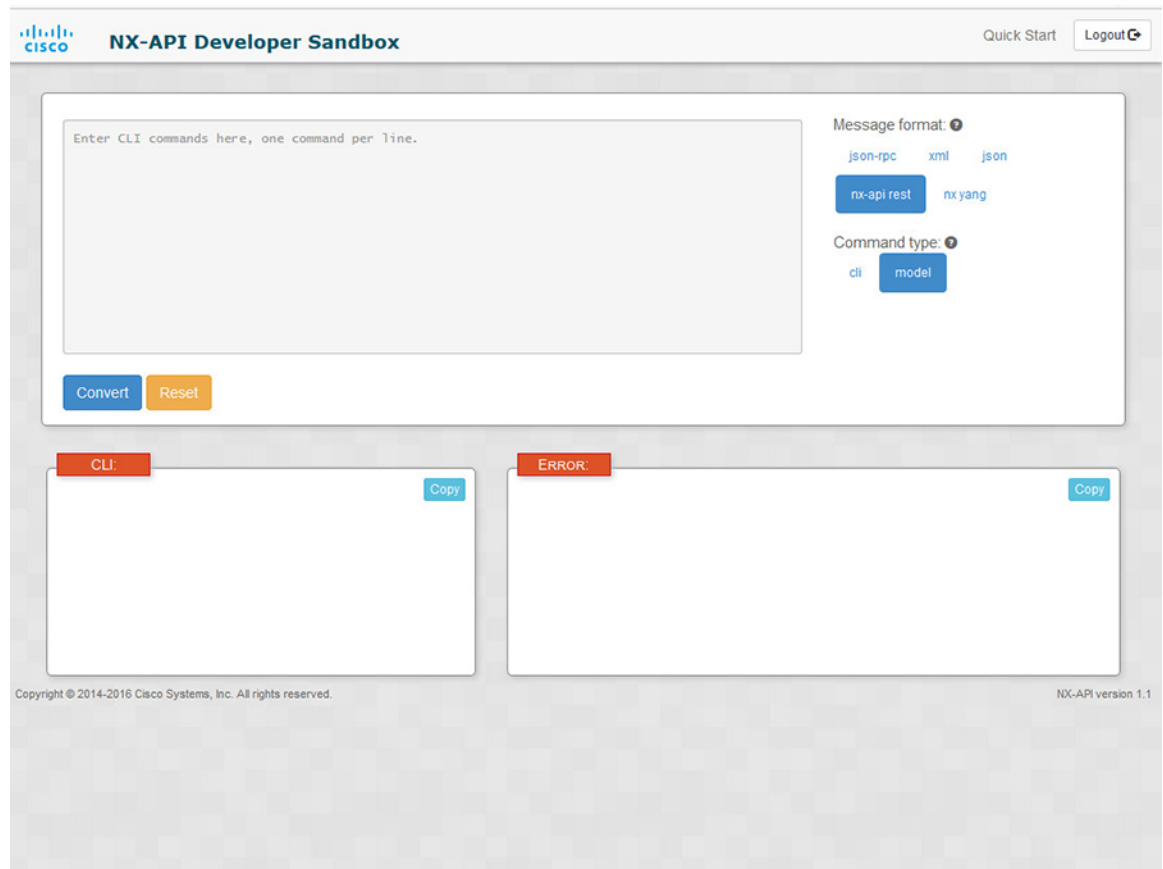
**Tip** Online help is available by clicking **Quick Start** in the upper right corner of the Sandbox window. Additional details, such as response codes and security methods, can be found in the NX-API CLI chapter. Only configuration commands are supported.

---

#### Procedure

---

- Step 1** Configure the **Message Format** and **Command Type** for the API protocol you want to use. For detailed instructions, see [Configuring the Message Format and Command Type, on page 180](#).
- Step 2** Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box in the top pane. You can erase the contents of the text entry box (and the **Request** and **Response** panes) by clicking **Reset** at the bottom of the top pane.



**Step 3** Click the **Convert** at the bottom of the top pane.

If the CLI commands contain no configuration errors, the payload appears in the **Request** pane. If errors are present, a descriptive error message appears in the **Response** pane.

The screenshot displays the NX-API Developer Sandbox interface. At the top, the Cisco logo and 'NX-API Developer Sandbox' are visible, along with 'Quick Start' and 'Logout' links. The main workspace is divided into several sections:

- Request Pane:** Contains a JSON payload:
 

```
api/mo/sys.json
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```
- Message format:** A dropdown menu with options: json-rpc, xml, json. The 'json' option is selected.
- Command type:** A dropdown menu with options: cli, model. The 'model' option is selected.
- Buttons:** 'Convert' (blue) and 'Reset' (orange) buttons are located below the request pane.
- Response Pane:** An empty area for displaying the switch's response.
- CLI Pane:** Shows the converted CLI command: 'hostname REST2CLI'. A 'Copy' button is present.
- ERROR Pane:** An empty area for displaying any error messages. A 'Copy' button is present.

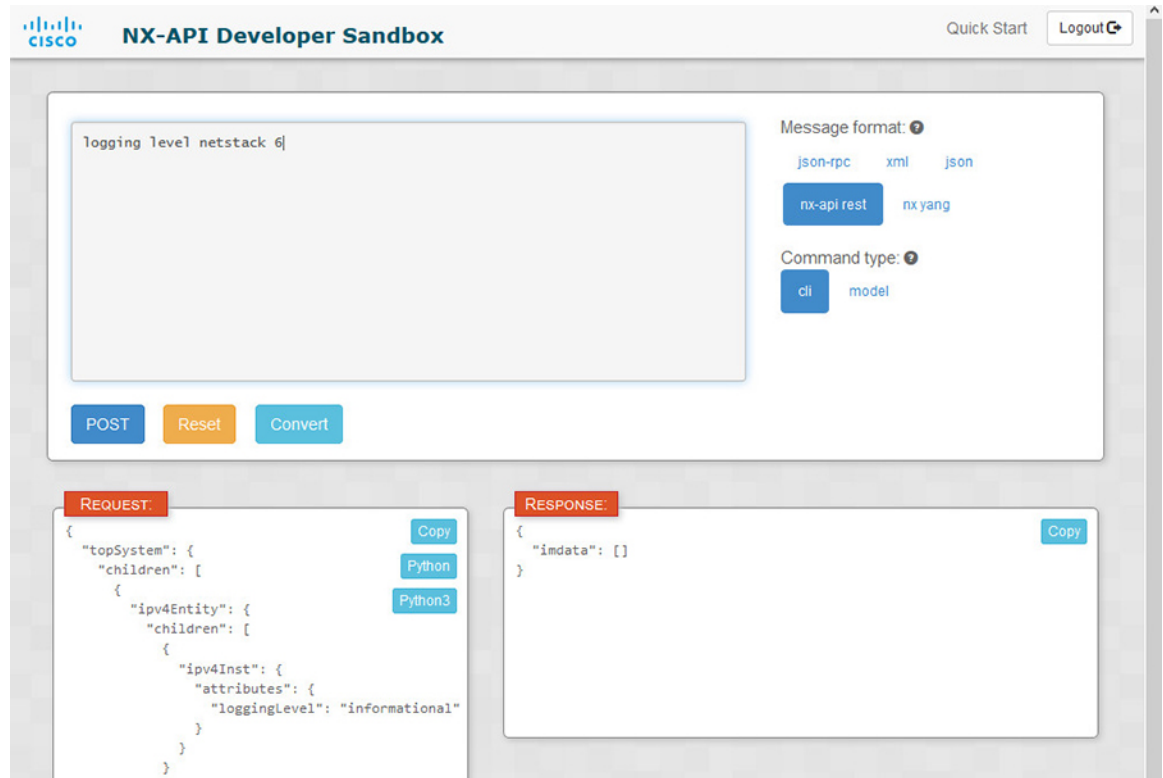
At the bottom of the interface, there is a status bar with the text 'Waiting for bam.nr-data.net...' and copyright information: 'Copyright © 2014-2016 Cisco Systems, Inc. All rights reserved.' and 'NX-API version 1.1'.

**Step 4** When a valid payload is present in the **Request** pane, you can click **POST** to send the payload as an API call to the switch.

The response from the switch appears in the **Response** pane.

**Warning** Clicking **POST** commits the command to the switch, which can result in a configuration or state change.





- Step 5** You can copy the contents of the **Request** or **Response** pane to the clipboard by clicking **Copy** in the pane.
- Step 6** You can obtain a Python implementation of the request on the clipboard by clicking **Python** in the **Request** pane.

## NX-API Developer Sandbox: NX-OS Release 9.2(2) and Later

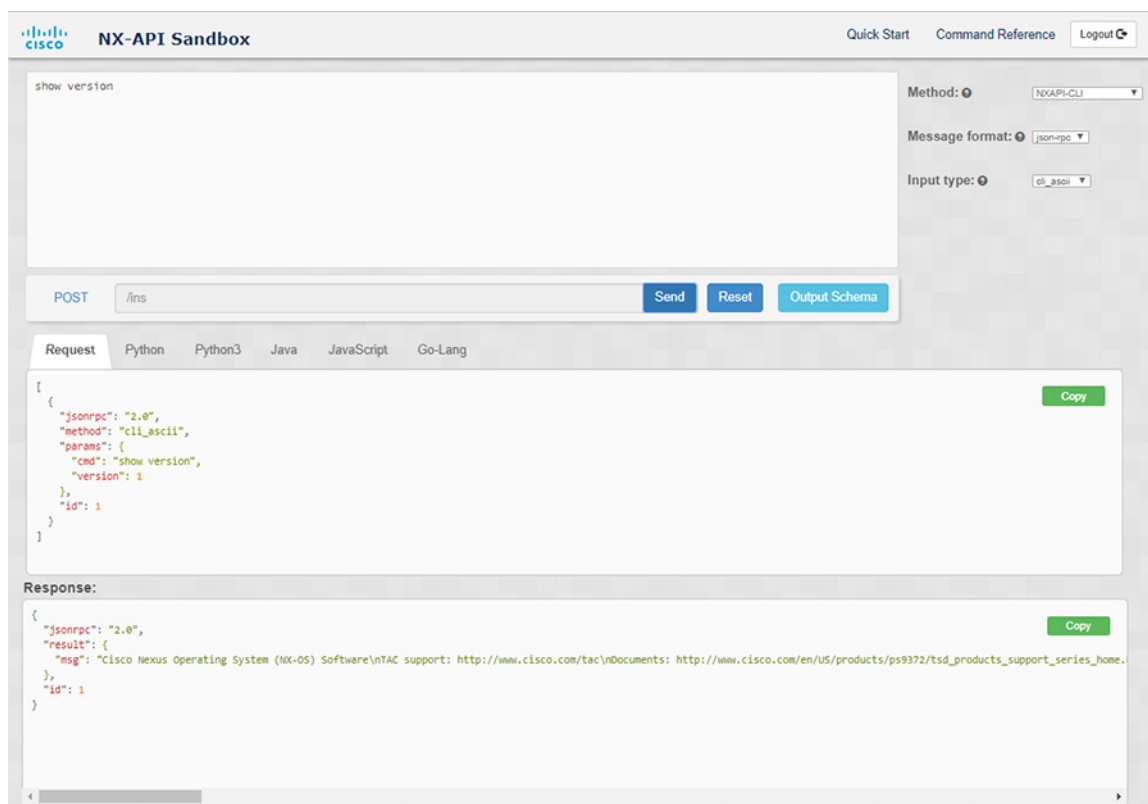
### About the NX-API Developer Sandbox

The Cisco NX-API Developer Sandbox is a web form hosted on the switch. It translates NX-OS CLI commands into equivalent XML or JSON payloads and converts NX-API REST payloads into their CLI equivalents.

The web form is a single screen with three panes — Command (top pane), Request (middle pane), and Response (bottom pane) — as shown in the figure below. The designated name (DN) field is located between the Command and Request panes (seen in the figure below located between the **POST** and **Send** options).

The Request pane also has a series of tabs. Each tab represents a different language: **Python**, **Python3**, **Java**, **JavaScript**, and **Go-Lang**. Each tab enables you to view the request in the respective language. For example, after converting CLI commands into an XML or JSON payload, click the **Python** tab to view the request in Python, which you can use to create scripts.

Figure 2: NX-API Developer Sandbox with Example Request and Output Response



Controls in the Command pane enable you to choose a supported API, such as NX-API REST, an input type, such as model (payload) or CLI, and a message format, such as XML or JSON. The available options vary depending on the chosen method.

When you choose the NX-API-REST (DME) method, type or paste one or more CLI commands into the Command pane, and click **Convert**, the web form converts the commands into a REST API payload, checking for configuration errors, and displays the resulting payload in the Request pane. If you then choose to post the payload directly from the sandbox to the switch (by choosing the **POST** option and clicking **SEND**), the Response pane displays the API response. For more information, see [Using the Developer Sandbox to Convert CLI Commands to REST Payloads, on page 189](#)

Conversely, the Cisco NX-API Developer Sandbox checks the payload for configuration errors then displays the equivalent CLIs in the Response pane. For more information, see [Using the Developer Sandbox to Convert from REST Payloads to CLI Commands, on page 192](#)

## Guidelines and Limitations

Following are the guidelines and limitations for the Developer Sandbox:

- Clicking **Send** in the Sandbox commits the command to the switch, which can result in a configuration or state change.
- Some feature configuration commands are not available until their associated feature has been enabled.

- Using Sandbox to convert with DN is supported only for finding the DN of a CLI config. Any other workflow, for example, using DME to convert DN for CLI configuration commands is not supported.
- The Command pane (the top pane) supports a maximum of 10,000 individual lines of input.

## Configuring the Message Format and Input Type

The **Method**, **Message format**, and **Input type** are configured in the upper right corner of the Command pane (the top pane). For **Method**, choose the format of the API protocol that you want to use. The Cisco NX-API Developer Sandbox supports the following API protocols:

*Table 11: NX-OS API Protocols*

Protocol	Description
NXAPI-CLI	Cisco NX-API proprietary protocol for delivering NX-OS CLI or bash commands in an XML or a JSON payload.
NXAPI-REST (DME)	<p>Cisco NX-API proprietary protocol for manipulating and reading managed objects (MOs) and their properties in the internal NX-OS data management engine (DME) model. The NXAPI-REST (DME) protocol displays a drop-down list that enables you to choose from the following methods:</p> <ul style="list-style-type: none"> <li>• <b>POST</b></li> <li>• <b>GET</b></li> <li>• <b>PUT</b></li> <li>• <b>DELETE</b></li> </ul> <p>For more information about the Cisco Nexus 3000 and 9000 Series NX-API REST SDK, see <a href="https://developer.cisco.com/site/cisco-nexus-nx-api-references/">https://developer.cisco.com/site/cisco-nexus-nx-api-references/</a>.</p>
RESTCONF (Yang)	<p>The YANG ("Yet Another Next Generation") data modeling language for configuration and state data.</p> <p>The RESTCONF (Yang) protocol displays a drop-down list that enables you to choose from the following methods:</p> <ul style="list-style-type: none"> <li>• <b>POST</b></li> <li>• <b>GET</b></li> <li>• <b>PUT</b></li> <li>• <b>PATCH</b></li> <li>• <b>DELETE</b></li> </ul>

When you choose the **Method**, a set of **Message format** or **Input type** options are displayed in a drop-down list. The **Message format** can constrain the input CLI and determine the **Request** and **Response** format. The options vary depending on the **Method** you choose.

The following table describes the **Input/Command type** options for each **Message format**:

Table 12: Command Types

Method	Message format	Input/Command type
NXAPI-CLI	json-rpc	<ul style="list-style-type: none"> <li>cli — show or configuration commands</li> <li>cli-ascii — show or configuration commands, output without formatting</li> <li>cli-array — show commands. Similar to cli, but with cli_array, data is returned as a list of one element, or an array, within square brackets, [ ].</li> </ul>
NXAPI-CLI	xml	<ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b> The bash shell must be enabled in the switch.</p>
NXAPI-CLI	json	<ul style="list-style-type: none"> <li>cli_show — show commands. If the command does not support XML output, an error message will be returned.</li> <li>cli_show_array — show commands. Similar to cli_show, but with cli_show_array, data is returned as a list of one element, or an array, within square brackets [ ].</li> <li>cli_show_ascii — show commands, output without formatting</li> <li>cli_conf — configuration commands. Interactive configuration commands are not supported.</li> <li>bash — bash commands. Most non-interactive bash commands are supported.</li> </ul> <p><b>Note</b> The bash shell must be enabled in the switch.</p>
NXAPI-REST (DME)		<ul style="list-style-type: none"> <li>cli — CLI to model conversion</li> <li>model — Model to CLI conversion.</li> </ul>

Method	Message format	Input/Command type
RESTCONF (Yang)	<ul style="list-style-type: none"> <li>• json — JSON structure is used for payload</li> <li>• xml — XML structure is used for payload</li> </ul>	

### Output Chunking

In order to handle large show command output, some NX-API message formats support output chunking for show commands. In this case, an **Enable chunk mode** check box appears below the **Command Type** control along with a session ID (**SID**) type-in box.

When chunking is enabled, the response is sent in multiple "chunks," with the first chunk sent in the immediate command response. In order to retrieve the next chunk of the response message, you must send an NX-API request with **SID** set to the session ID of the previous response message.

## Using the Developer Sandbox

You can use the Cisco NX-API Developer Sandbox to make multiple conversions, including the following:

### Using the Developer Sandbox to Convert CLI Commands to REST Payloads



#### Tip

- Online help is available by clicking the help icons (?) next to the field names located in the upper-right corner of the Cisco NX-API Developer Sandbox window.
- For additional details, such as response codes and security methods, see the *NX-API CLI* chapter.
- Only configuration commands are supported.

The Cisco NX-API Developer Sandbox enables you to convert CLI commands to REST payloads.

#### Procedure

- Step 1** Click the **Method** drop-down list and choose **NXAPI-REST (DME)**.  
The **Input** type drop-down list appears.
- Step 2** Click the **Input** type drop-down list and choose **cli**.
- Step 3** Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box in the top pane.  
You can erase the contents of the text entry box (and the **Request** and **Response** panes) by clicking **Reset** at the bottom of the top pane.

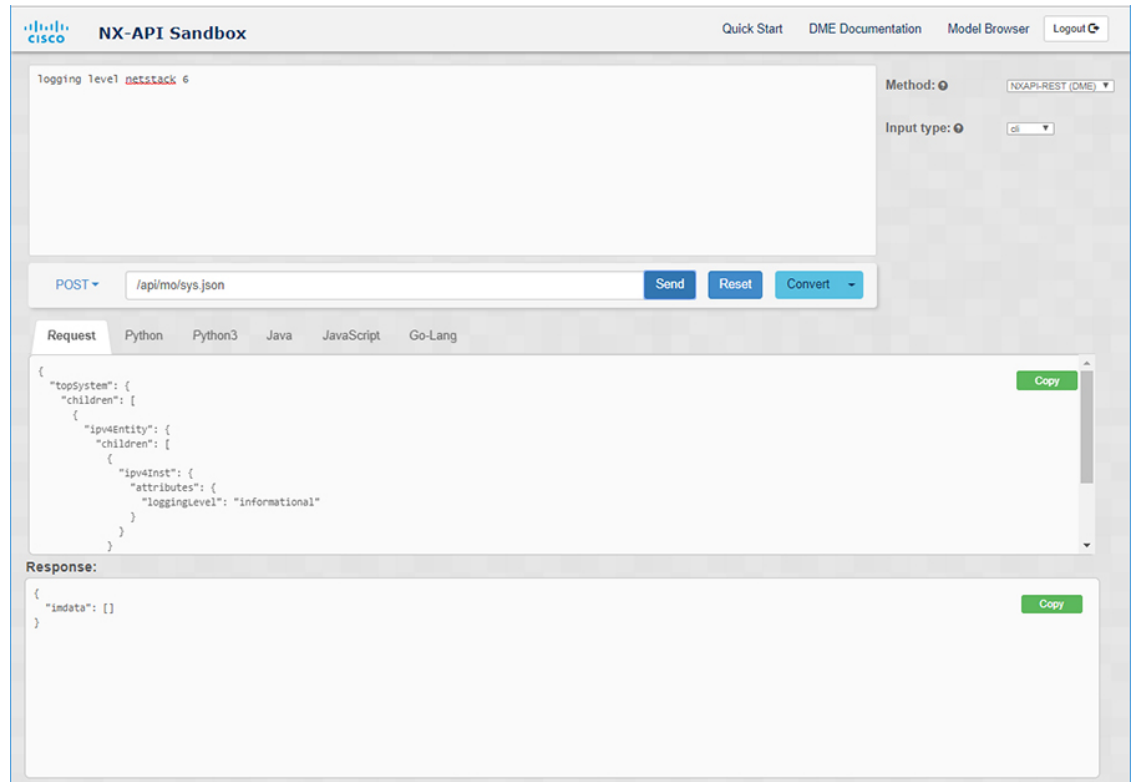
**Step 4** Click **Convert**.

If the CLI commands contain no configuration errors, the payload appears in the **Request** pane. If errors are present, a descriptive error message appears in the **Response** pane.

**Step 5** (Optional) To send a valid payload as an API call to the switch, click **Send**.

The response from the switch appears in the **Response** pane.

**Warning** Clicking **Send** commits the command to the switch, which can result in a configuration or state change.



**Step 6** (Optional) To obtain the DN for an MO in the payload:

- a. From the **Request** pane, choose **POST**.
- b. Click the **Convert** drop-down list and choose **Convert (with DN)**.

The payload appears with a **dn** field that contains the DN that corresponds to each MO in the payload.

**Step 7** (Optional) To overwrite the current configuration with a new configuration:

- a. Click the **Convert** drop-down list and choose **Convert (for Replace)**. The **Request** pane displays a payload with a **status** field set to **replace**.
- b. From the **Request** pane, choose **POST**.
- c. Click **Send**.

The current configuration is replaced with the posted configuration. For example, if you start with the following configuration:

```
interface eth1/2
  description test
  mtu 1501
```

Then use **Convert (for Replace)** to POST the following configuration:

```
interface eth1/2
  description testForcr
```

The `mtu` configuration is removed and only the new description (`testForcr`) is present under the interface. This change is confirmed when entering **show running-config**.

**Step 8** (Optional) To copy the contents of a pane, such as the **Request** or **Response** pane, click **Copy**. The contents of the respective pane is copied to the clipboard.

**Step 9** (Optional) To convert the request into an of the formats listed below, click on the appropriate tab in the **Request** pane:

- **Python**
- **Python3**
- **Java**
- **JavaScript**
- **Go-Lang**

---

## Using the Developer Sandbox to Convert from REST Payloads to CLI Commands

The Cisco NX-API Developer Sandbox enables you to convert REST payloads to corresponding CLI commands. This option is only available for the NXAPI-REST (DME) method.



### Tip

- Online help is available by clicking help icons (?) next to the Cisco NX-API Developer Sandbox field names. Click a help icon get information about the respective field.

For additional details, such as response codes and security methods, see the chapter *NX-API CLI*.

- The top-right corner of the Cisco NX-API Developer Sandbox contains links for additional information. The links that appear depend on the **Method** you choose. The links that appear for the NXAPI-REST (DME) method:

- **NX-API References**—Enables you to access additional NX-API documentation.
- **DME Documentation**—Enables you to access the NX-API DME Model Reference page.
- **Model Browser**—Enables you to access Visore, the Model Browser. Note that you might have to manually enter the IP address for your switch to access the Visore page:

`https://management-ip-address/visore.html`.

---

### Procedure

**Step 1** Click the **Method** drop-down list and choose **NXAPI-REST (DME)**.

**Example:**



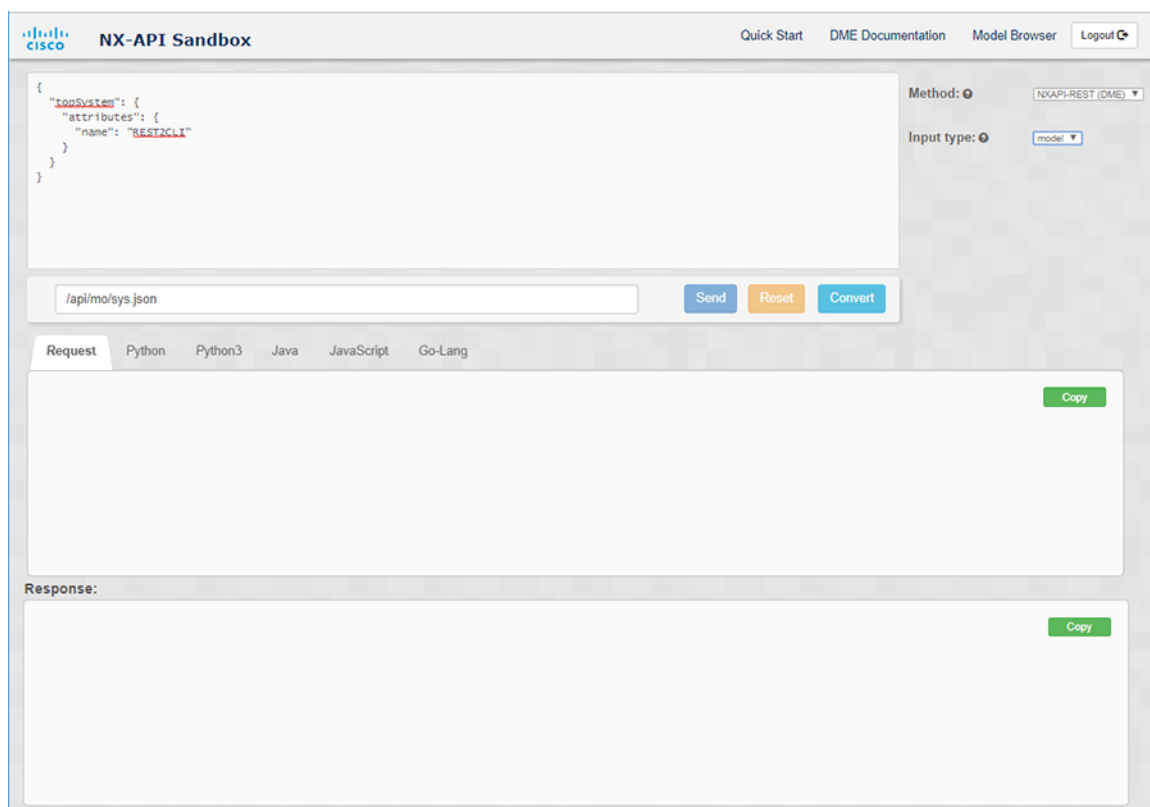
The screenshot shows the NX-API Sandbox interface. At the top, there is a navigation bar with the Cisco logo, the title "NX-API Sandbox", and links for "Quick Start", "DME Documentation", "Model Browser", and "Logout". Below the navigation bar is a large text area for entering the DME payload. To the right of this area are two dropdown menus: "Method" set to "NX-API-REST (DME)" and "Input type" set to "model". Below the text area is a text input field containing the DN "/api/mo/sys.json", followed by "Send", "Reset", and "Convert" buttons. Below the input field is a tabbed interface with "Request" selected, and other tabs for "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab shows a large empty text area with a "Copy" button. Below the "Request" tab is a "Response:" section, also with a large empty text area and a "Copy" button.

- Step 2** Click the **Input Type** drop-down list and choose **model**.
- Step 3** Enter the designated name (DN) that corresponds to the payload in the field above the Request pane.
- Step 4** Enter the payload in the Command pane.
- Step 5** Click **Convert**.

**Example:**

For this example, the DN is `/api/mo/sys.json` and the NX-API REST payload is:

```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```



When you click on the **Convert** button, the CLI equivalent appears in the **CLI** pane as shown in the following image.

**NX-API Sandbox**

[Quick Start](#)
[DME Documentation](#)
[Model Browser](#)
[Logout](#)

```
{
  "topSystem": {
    "attributes": {
      "name": "REST2CLI"
    }
  }
}
```

Send
Reset
Convert

Request
Python
Python3
Java
JavaScript
Go-Lang

```
hostname REST2CLI
```

Copy

**Response:**

Copy

**Note** The Cisco NX-API Developer Sandbox cannot convert all payloads into equivalent CLIs, even if the sandbox converted the CLIs to NX-API REST payloads. The following is a list of possible sources of error that can prevent a payload from completely converting to CLI commands:

**Table 13: Sources of REST2CLI Errors**

Payload Issue	Result
<p>The payload contains an attribute that does not exist in the MO.</p> <p>Example:</p> <pre>api/mo/sys.json {   "topSystem": {     "children": [       {         "interfaceEntity": {           "children": [             {               "l1PhysIf": {                 "attributes": {                   "id": "eth1/1",                   "fakeattribute": "totallyFake"                 }               }             }           ]         }       }     ]   } }</pre>	<p>The <b>Error</b> pane will return an error related to the attribute.</p> <p>Example:</p> <p><b>CLI</b></p> <p><b>Error</b> unknown attribute 'fakeattribute' in element 'l1PhysIf'</p>
<p>The payload includes MOs that aren't yet supported for conversion:</p> <p>Example:</p> <pre>api/mo/sys.json {   "topSystem": {     "children": [       {         "dhcpEntity": {           "children": [             {               "dhcpInst": {                 "attributes": {                   "SnoopingEnabled": "yes"                 }               }             }           ]         }       }     ]   } }</pre>	<p>The <b>Error</b> Pane will return an error related to the unsupported MO.</p> <p>Example:</p> <p><b>CLI</b></p> <p><b>Error</b> The entire subtree of "sys/dhcp" is not converted.</p>

## Using the Developer Sandbox to Convert from RESTCONF to json or XML

**Tip**

- Online help is available by clicking the help icon (?) in the upper-right corner of the Cisco NX-API Developer Sandbox window.
- Click on the **Yang Documentation** link in the upper right corner of the Sandbox window to go to the Model Driven Programmability with Yang page.
- Click on the **Yang Models** link in the upper right corner of the Sandbox window to access the YangModels GitHub site.

### Procedure

**Step 1** Click the **Method** drop-down list and choose **RESTCONF (Yang)**.

**Example:**

The screenshot displays the Cisco NX-API Sandbox interface. At the top, there are navigation links for 'Quick Start', 'Yang Documentation', 'Yang Models', and 'Logout'. The main area is divided into several sections:

- Logging level:** A text input field containing 'netstack'.
- Method:** A dropdown menu currently set to 'RESTCONF (Yang)'.
- Message format:** A dropdown menu currently set to 'json'.
- Request:** A section with a dropdown menu set to 'POST' and a text input field containing the URL 'restconf/data/Cisco-NX-OS-device:System/'. Below the input are three buttons: 'Send' (orange), 'Reset' (blue), and 'Convert' (blue).
- Response:** A section with a text input field and a 'Copy' button (green).

**Step 2** Click **Message format** and choose either **json** or **xml**.

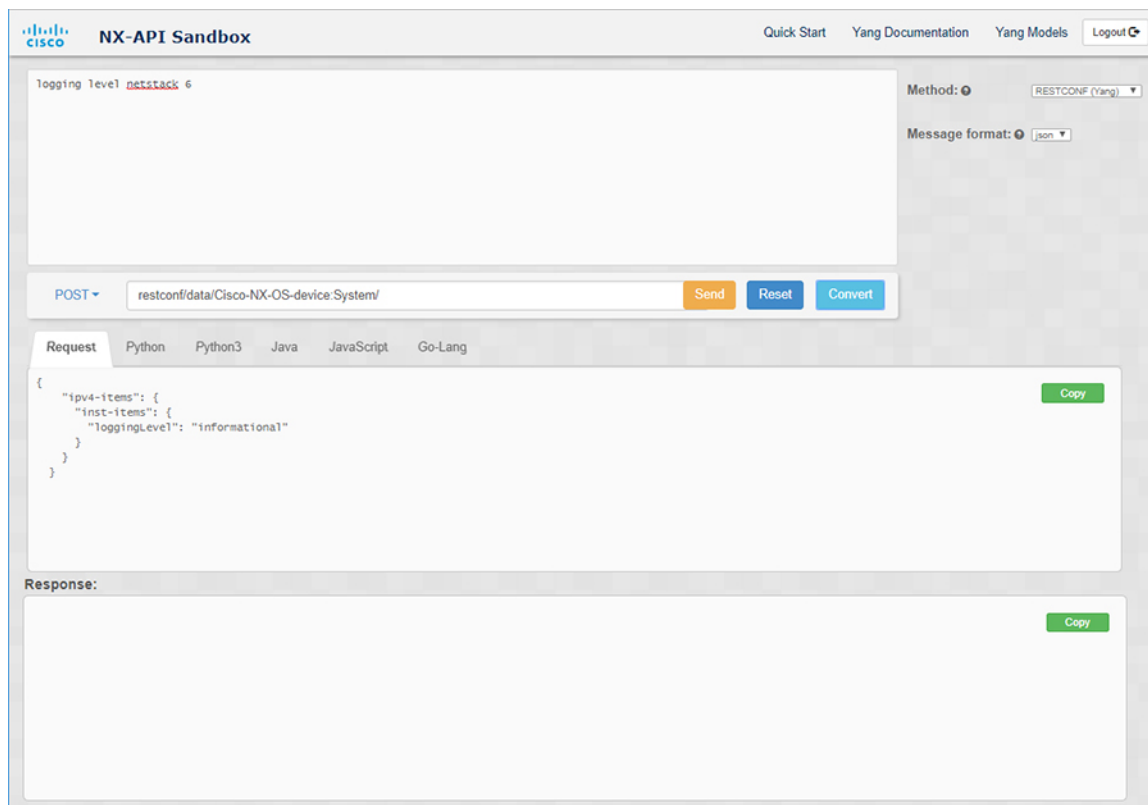
**Step 3** Enter a command in the text entry box in the top pane.

**Step 4** Choose a message format.

**Step 5** Click **Convert**.

**Example:**

For this example, the command is **logging level netstack 6** and the message format is json:



The screenshot displays the Cisco NX-API Sandbox interface. At the top, the Cisco logo and "NX-API Sandbox" are visible, along with navigation links for "Quick Start", "Yang Documentation", "Yang Models", and "Logout". The main area contains a text input field with the command "logging level netstack 6". To the right, there are dropdown menus for "Method" (set to "RESTCONF (Yang)") and "Message format" (set to "json"). Below the input field, a "POST" dropdown is followed by a text box containing the URL "restconf/data/Cisco-NX-OS-device:System/". Three buttons are present: "Send" (orange), "Reset" (blue), and "Convert" (blue). Underneath, there are tabs for "Request", "Python", "Python3", "Java", "JavaScript", and "Go-Lang". The "Request" tab is active, showing a JSON object: 

```
{  "ipv4-items": {    "inst-items": {      "loggingLevel": "informational"    }  } }
```

 A green "Copy" button is located to the right of the JSON. Below the request, there is a "Response:" label and an empty text area with a green "Copy" button to its right.

**Example:**

For this example, the command is **logging level netstack 6** and the message format is xml:

The screenshot shows the Cisco NX-API Sandbox interface. At the top, there is a header with the Cisco logo, 'NX-API Sandbox', and navigation links for 'Quick Start', 'Yang Documentation', 'Yang Models', and 'Logout'. The main area is divided into two sections. The top section contains a text input field with the text 'logging level **netstack** 6'. To the right of this field are two dropdown menus: 'Method:' set to 'RESTCONF (Yang)' and 'Message format:' set to 'xml'. Below the input field is a 'POST' dropdown menu and a text input field containing the URL 'restconf/data/Cisco-NX-OS-device:System/'. To the right of the URL are three buttons: 'Send' (orange), 'Reset' (blue), and 'Convert' (blue). Below the URL field is a 'Request' tab, which is currently selected. Underneath the 'Request' tab are five sub-tabs: 'Python', 'Python3', 'Java', 'JavaScript', and 'Go-Lang'. The 'Request' tab displays the following XML content: 

```
<ipv4-items>
<inst-items>
  <loggingLevel>informational</loggingLevel>
</inst-items>
</ipv4-items>
```

 To the right of the XML content is a green 'Copy' button. Below the 'Request' section is a 'Response:' section, which is currently empty. To the right of the 'Response:' section is another green 'Copy' button.

**Step 6** You can also convert the request into the following formats by clicking on the appropriate tab in the **Request** pane:

- Python
- Python3
- Java
- JavaScript
- Go-Lang

**Note** The Java-generated script does not work if you choose the PATCH option from the drop-down menu in the area above the Request tab. This is a known limitation with Java and is expected behavior.







## PART **IV**

# Model-Driven Programmability

- [Managing Components, on page 203](#)
- [Converting CLI Commands to Network Configuration Format, on page 209](#)





## CHAPTER 20

# Managing Components

- [About the Component RPM Packages, on page 203](#)
- [Preparing For Installation, on page 205](#)
- [Downloading Components from the Cisco Artifactory, on page 206](#)
- [Installing RPM Packages, on page 207](#)

## About the Component RPM Packages



**Note** Beginning with Cisco Nexus NX-OS 7.0(3)I6(2), the NX-OS Programmable Interface Base Component RPM packages (agents, the Cisco native model, most of the other required models, and infrastructure) are included in the NX-OS image. As a result, nearly all the required software is installed automatically when the image is loaded. This situation means that there is no need to download and install the bulk of the software from the Cisco Artifactory. The exception is the OpenConfig model, which is required. You must explicitly download the OpenConfig models from the Cisco Artifactory.

But, for Cisco Nexus NX-OS 7.0(3)I6(1) and earlier releases, if you need to upgrade, the following sections describing downloading and installing the packages are required.

NX-OS Programmable Interface Component RPM packages may be downloaded from the Cisco Artifactory. There are two types of component RPM packages that are needed:

- Base Components (required)
- Common Model Components (OpenConfig models must be explicitly downloaded and installed)

### Base Components

The Base Components comprise the following required RPM packages:

- **mtx-infra** — Infrastructure
- **mtx-device** — Cisco native model

At least one of the following agent packages must be installed in order to have access to the modeled NX-OS interface:

- **mtx-netconf-agent** — NETCONF agent

- **mtx-restconf-agent** — RESTCONF agent
- **mtx-grpc-agent** — gRPC agent

### Common Model Components

Common Model component RPMs support OpenConfig models. To use the OpenConfig models, you must download and install the OpenConfig RPMs. For convenience, there is a single combined package of all supported OpenConfig models, `mtx-openconfig-all`.

While the single combined package is recommended, an alternative is to download and install RPMs of selected models and their dependencies among the supported models listed in the following table. The `mtx-openconfig-all` RPM is not compatible with the individual model RPMs. You must uninstall the former before installing the latter, and you must uninstall the latter before installing the former.

Model Name	Model Rev	Model Ver	Package Name	Dependencies
openconfig-acl	2017-05-26	1.0.0	mtx-openconfig-acl	mtx-openconfig-interfaces
openconfig-bgp-policy	2017-07-30	4.0.1	mtx-openconfig-bgp-policy	mtx-openconfig-interfaces mtx-openconfig-routing-policy
openconfig-if-aggregate	2017-07-14	2.0.0	mtx-openconfig-if-aggregate	mtx-openconfig-if-ethernet mtx-openconfig-interfaces
openconfig-if-ethernet	2017-07-14	2.0.0	mtx-openconfig-if-ethernet	mtx-openconfig-interfaces
openconfig-if-ip	2016-05-26	1.0.2	mtx-openconfig-if-ip	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-vlan
openconfig-if-ip-ext	2018-01-05	2.3.0	mtx-openconfig-if-ip-ext	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-if-ip mtx-openconfig-interfaces mtx-openconfig-vlan
openconfig-interfaces	2017-07-14	2.0.0	mtx-openconfig-interfaces	-

Model Name	Model Rev	Model Ver	Package Name	Dependencies
openconfig-network-instance	2017-08-24	0.8.1	mtx-openconfig-network-instance	mtx-openconfig-bgp-policy mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-routing-policy mtx-openconfig-vlan
openconfig-network-instance-policy	2017-02-15	0.1.0	mtx-openconfig-network-instance-policy	mtx-openconfig-routing-policy
openconfig-ospf-policy	2017-08-24	0.1.1	mtx-openconfig-ospf-policy	mtx-openconfig-interfaces mtx-openconfig-routing-policy
openconfig-platform	2018-01-16	0.8.0	mtx-openconfig-platform	-
openconfig-platform-linecard	2017-08-03	0.1.0	mtx-openconfig-platform-linecard	mtx-openconfig-platform
openconfig-platform-port	2018-01-20	0.3.0	mtx-openconfig-platform-port	mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-platform
openconfig-platform-transceiver	2018-01-22	0.4.1	mtx-openconfig-platform-transceiver	mtx-openconfig-if-ethernet mtx-openconfig-interfaces mtx-openconfig-platform
openconfig-relay-agent	2016-05-16	0.1.0	mtx-openconfig-relay-agent	mtx-openconfig-interfaces
openconfig-routing-policy	2016-05-12	2.0.1	mtx-openconfig-routing-policy	-
openconfig-spanning-tree	2017-07-14	0.2.0	mtx-openconfig-spanning-tree	mtx-openconfig-interfaces
openconfig-system	2017-09-18	0.3.0	mtx-openconfig-system	-
openconfig-vlan	2017-07-14	2.0.0	mtx-openconfig-vlan	mtx-openconfig-if-aggregate mtx-openconfig-if-ethernet mtx-openconfig-interfaces

## Preparing For Installation

This section contains installation preparation and other useful information for managing NX-OS Programmable Interface components.

### Opening the Bash Shell on the Device

RPM installation on the switch is performed in the Bash shell. Make sure that **feature bash** is configured on the device.

```
Switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# feature bash-shell
Switch(config)# end
Switch# run bash sudo su
bash-4.2#
```

To return to the device CLI prompt from Bash, type **exit** or **Ctrl-D**.

### Verify Device Readiness

You can use the following CLI **show** commands to confirm the readiness of the device before installation of an RPM.

- `show module` — Indicates whether all modules are up.
 

```
Switch# show module
```
- `show system redundancy status` — Indicates whether the standby device is up and running and in HA mode. If a standby sync is in progress, the RPM installation may fail.
 

```
Switch# show system redundancy status
```

If the line cards have failed to come up, enter the `createrepo /rpms` command in the Bash shell.

```
bash-4.2# createrepo /rpms
```

## Downloading Components from the Cisco Artifacts

The NX-OS Programmable Interface Component RPMs can be downloaded from the Cisco Artifacts at the following URL. The RPMs are organized by NX-OS release-specific directories. Ensure that you are downloading the RPMs from the correct NX-OS release directory.

<https://devhub.cisco.com/artifacts/open-nxos-agents>

The NX-OS Programmable Interface Component RPMs adhere to the following naming convention:

`<package>-<version>-<NX-OS release>.<architecture>.rpm`

Select and download the desired NX-OS Programmable Interface Component RPM packages to the device for installation as described in the following sections.

# Installing RPM Packages

## Installing the Programmable Interface Base And Common Model Component RPM Packages

### Before you begin

- From the Cisco Artifacts, download the following packages:
  - mtz-infra
  - mtz-device
  - mtz-netconf-agent/mtz-restconf-agent/mtz-grpc-agent (at least one)
  - mtz-openconfig-all (alternatively, selected individual models)
- Using the CLI commands in [Verify Device Readiness, on page 206](#), confirm that all line cards in the Active and Standby devices are up and ready.

### Procedure

---

**Step 1** Copy the downloaded RPMs to the device.

#### Example:

```
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtz-infra-2.0.0.0-9.2.1.lib32_n9000.rpm bootflash:
vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtz-device-2.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtz-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
Switch# copy scp://jdoe@192.0.20.123/myrpms/mtz-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
bootflash: vrf management
```

**Step 2** From the Bash shell, install the RPMs.

#### Example:

```
bash-4.2# cd /bootflash
bash-4.2# yum install mtz-infra-2.0.0.0-9.2.1.lib32_n9000.rpm
mtz-device-2.0.0.0-9.2.1.lib32_n9000.rpm mtz-netconf-agent-2.0.0.0-9.2.1.lib32_n9000.rpm
mtz-openconfig-all-1.0.0.0-9.2.1.lib32_n9000.rpm
```

**Step 3** From the Bash shell, verify the installation.

#### Example:

```
bash-4.2# yum list installed | grep mt*
```

---





## CHAPTER 21

# Converting CLI Commands to Network Configuration Format

- [Information About XMLIN, on page 209](#)
- [Licensing Requirements for XMLIN, on page 209](#)
- [Installing and Using the XMLIN Tool, on page 210](#)
- [Converting Show Command Output to XML, on page 210](#)
- [Configuration Examples for XMLIN, on page 211](#)

## Information About XMLIN

The XMLIN tool converts CLI commands to the Network Configuration (NETCONF) protocol format. NETCONF is a network management protocol that provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses XML-based encoding for configuration data and protocol messages. The NX-OS implementation of the NETCONF protocol supports the following protocol operations: `<get>`, `<edit-config>`, `<close-session>`, `<kill-session>`, and `<exec-command>`.

The XMLIN tool converts show, EXEC, and configuration commands to corresponding NETCONF `<get>`, `<exec-command>`, and `<edit-config>` requests. You can enter multiple configuration commands into a single NETCONF `<edit-config>` instance.

The XMLIN tool also converts the output of show commands to XML format.

## Licensing Requirements for XMLIN

**Table 14: XMLIN Licensing Requirements**

Product	License Requirement
Cisco NX-OS	XMLIN requires no license. Any feature not included in a license package is bundled with the Cisco NX-OS system images and is provided at no extra charge to you. For a complete explanation of the Cisco NX-OS licensing scheme, see the <i>Cisco NX-OS Licensing Guide</i> .

# Installing and Using the XMLIN Tool

You can install the XMLIN tool and then use it to convert configuration commands to NETCONF format.

## Before you begin

The XMLIN tool can generate NETCONF instances of commands even if the corresponding feature sets or required hardware capabilities are not available on the device. But, you might still need to install some feature sets before entering the **xmlin** command.

## Procedure

	Command or Action	Purpose
<b>Step 1</b>	switch# <b>xmlin</b>	
<b>Step 2</b>	switch(xmlin)# <b>configure terminal</b>	Enters global configuration mode.
<b>Step 3</b>	Configuration commands	Converts configuration commands to NETCONF format.
<b>Step 4</b>	(Optional) switch(config)(xmlin)# <b>end</b>	Generates the corresponding <edit-config> request.  <b>Note</b> Enter the <b>end</b> command to finish the current XML configuration before you generate an XML instance for a <b>show</b> command.
<b>Step 5</b>	(Optional) switch(config-if-verify)(xmlin)# <b>show commands</b>	Converts <b>show</b> commands to NETCONF format.
<b>Step 6</b>	(Optional) switch(config-if-verify)(xmlin)# <b>exit</b>	Returns to EXEC mode.

# Converting Show Command Output to XML

You can convert the output of show commands to XML.

## Before you begin

Make sure that all features for the commands you want to convert are installed and enabled on the device. Otherwise, the commands fail.

You can use the **terminal verify-only** command to verify that a feature is enabled without entering it on the device.

Make sure that all required hardware for the commands you want to convert are present on the device. Otherwise, the commands fail.

Make sure that the XMLIN tool is installed.

**Procedure**

	Command or Action	Purpose
<b>Step 1</b>	switch# <i>show-command</i>   <b>xmlin</b>	Enters global configuration mode.  <b>Note</b> You cannot use this command with configuration commands.

## Configuration Examples for XMLIN

The following example shows how the XMLIN tool is installed on the device and used to convert a set of configuration commands to an <edit-config> instance.

```

switch# xmlin
*****
Loading the xmlin tool. Please be patient.
*****
Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Copyright ©) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under
license. Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or the GNU
Lesser General Public License (LGPL) Version 2.1. A copy of each
such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://www.opensource.org/licenses/lgpl-2.1.php

switch(xmlin)# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)(xmlin)# interface ethernet 2/1
% Success
switch(config-if-verify)(xmlin)# cdp enable
% Success
switch(config-if-verify)(xmlin)# end
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:configure_"
xmlns:m="http://www.cisco.com/nxos:6.2.2.:_exec"
xmlns:ml="http://www.cisco.com/nxos:6.2.2.:configure__if-eth-base" message-id="1">
  <nf:edit-config>
    <nf:target>
      <nf:running/>
    </nf:target>
    <nf:config>
      <m:configure>
        <m:terminal>
          <interface>
            <__XML_PARAM_interface>
              <__XML_value>Ethernet2/1</__XML_value>
              <ml:cdp>
                <ml:enable/>
              </ml:cdp>
            </__XML_PARAM_interface>
          </interface>
        </m:terminal>
      </m:configure>
    </nf:config>
  </nf:edit-config>
</nf:rpc>

```

```

    </nf:config>
  </nf:edit-config>
</nf:rpc>
]]>]]>

```

The following example shows how to enter the **end** command to finish the current XML configuration before you generate an XML instance for a **show** command.

```

switch(xmlin)# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)(xmlin)# interface ethernet 2/1
switch(config-if-verify)(xmlin)# show interface ethernet 2/1
*****
Please type "end" to finish and output the current XML document before building a new one.
*****
% Command not successful

switch(config-if-verify)(xmlin)# end
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:configure_"
xmlns:m="http://www.cisco.com/nxos:6.2.2.:_exec" message-id="1">
  <nf:edit-config>
    <nf:target>
      <nf:running/>
    </nf:target>
    <nf:config>
      <m:configure>
        <m:terminal>
          <interface>
            <__XML_PARAM__interface>
              <__XML_value>Ethernet2/1</__XML_value>
            </__XML_PARAM__interface>
          </interface>
        </m:terminal>
      </m:configure>
    </nf:config>
  </nf:edit-config>
</nf:rpc>
]]>]]>

switch(xmlin)# show interface ethernet 2/1
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:if_manager" message-id="1">
  <nf:get>
    <nf:filter type="subtree">
      <show>
        <interface>
          <__XML_PARAM__ifeth>
            <__XML_value>Ethernet2/1</__XML_value>
          </__XML_PARAM__ifeth>
        </interface>
      </show>
    </nf:filter>
  </nf:get>
</nf:rpc>
]]>]]>
switch(xmlin)# exit
switch#

```

The following example shows how you can convert the output of the **show interface brief** command to XML.

```
switch# show interface brief | xmlin
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:6.2.2.:if_manager"

message-id="1">
  <nf:get>
    <nf:filter type="subtree">
      <show>
        <interface>
          <brief/>
        </interface>
      </show>
    </nf:filter>
  </nf:get>
</nf:rpc>
]]>]]>
```





## PART **V**

# XML Management Interface

- [XML Management Interface, on page 217](#)







## CHAPTER 22

# XML Management Interface

---

This section contains the following topics:

- [About the XML Management Interface, on page 217](#)
- [Licensing Requirements for the XML Management Interface, on page 218](#)
- [Prerequisites to Using the XML Management Interface, on page 219](#)
- [Using the XML Management Interface, on page 219](#)
- [Information About Example XML Instances, on page 231](#)
- [Additional References, on page 237](#)

## About the XML Management Interface

### About the XML Management Interface

You can use the XML management interface to configure a device. The interface uses the XML-based Network Configuration Protocol (NETCONF), which allows you to manage devices and communicate over the interface with an XML management tool or program. The Cisco NX-OS implementation of NETCONF requires you to use a Secure Shell (SSH) session for communication with the device.

NETCONF is implemented with an XML Schema (XSD) that allows you to enclose device configuration elements within a remote procedure call (RPC) message. From within an RPC message, you select one of the NETCONF operations that matches the type of command that you want the device to execute. You can configure the entire set of CLI commands on the device with NETCONF. For information about using NETCONF, see the [Creating NETCONF XML Instances, on page 221](#) and [RFC 4741](#).

For more information about using NETCONF over SSH, see [RFC 4742](#).

This section includes the following topics:

- [NETCONF Layers, on page 217](#)
- [SSH xmlagent, on page 218](#)

### NETCONF Layers

The following are the NETCONF layers:

Table 15: NETCONF Layers

Layer	Example
Transport protocol	SSHv2
RPC	<rpc>, <rpc-reply>
Operations	<get-config>, <edit-config>
Content	show or configuration command

The following is a description of the four NETCONF layers:

- SSH transport protocol—Provides a secure, encrypted connection between a client and the server.
- RPC tag—Introduces a configuration command from the requestor and the corresponding reply from the XML server.
- NETCONF operation tag—Indicates the type of configuration command.
- Content—Indicates the XML representation of the feature that you want to configure.

## SSH xmlagent

The device software provides an SSH service that is called xmlagent that supports NETCONF over SSH Version 2.



### Note

The xmlagent service is referred to as the XML server in the Cisco NX-OS software.

NETCONF over SSH starts with the exchange of a hello message between the client and the XML server. After the initial exchange, the client sends XML requests, which the server responds to with XML responses. The client and server terminate requests and responses with the character sequence >. Because this character sequence is not valid in XML, the client and the server can interpret when the messages end, which keeps communication in sync.

The XML schemas that define XML configuration instances that you can use are described in the [Creating NETCONF XML Instances, on page 221](#) section.

## Licensing Requirements for the XML Management Interface

Product	Product
Cisco NX-OS	The XML management interface requires no license. Any feature not included in a license package is bundled with the Cisco NX-OS image and is provided at no extra charge to you. For a complete explanation of the Cisco NX-OS licensing scheme, see the <i>Cisco NX-OS Licensing Guide</i> .

# Prerequisites to Using the XML Management Interface

The XML management interface has the following prerequisites:

- You must install SSHv2 on the client PC.
- You must install an XML management tool that supports NETCONF over SSH on the client PC.
- You must set the appropriate options for the XML server on the device.

## Using the XML Management Interface

This section describes how to manually configure and use the XML management interface. Use the XML management interface with the default settings on the device.

### Configuring SSH and the XML Server Options

By default, the SSH server is enabled on the device. If you disable SSH, you must enable it before you start an SSH session on the client PC.

You can configure XML server options to control the number of concurrent sessions and the timeout for active sessions. You can also enable XML document validation and terminate XML sessions.



---

**Note** The XML server timeout applies only to active sessions.

---

For more information about configuring SSH, see the Cisco NX-OS security configuration guide for your platform.

For more information about the XML commands, see the Cisco NX-OS system management configuration guide for your platform.

### Starting an SSH Session

You can start an SSHv2 session on the client PC with a command similar to the following:

```
ssh2 username@ip-address -s xmlagent
```

Enter the login username, the IP address of the device, and the service to connect to. The xmlagent service is referred to as the XML server in the device software.



---

**Note** The SSH command syntax can differ from the SSH software on the client PC.

---

If you do not receive a hello message from the XML server, verify the following conditions:

- The SSH server is enabled on the device.
- The XML server max-sessions option is adequate to support the number of SSH connections to the device.

- The active XML server sessions on the device are not all in use.

## Sending the Hello Message

When you start an SSH session to the XML server, the server responds immediately with a hello message that informs the client of the server's capabilities. You must advertise your capabilities to the server with a hello message before the server processes any other requests. The XML server supports only base capabilities and expects support only for the base capabilities from the client.

The following are sample hello messages from the server and the client.



**Note** You must end all XML documents with `]]>]]>` to support synchronization in NETCONF over SSH.

### Hello Message from the server

```
<?xml version="1.0"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:xml:ns:netconf:base:1.0</capability>
  </capabilities>
  <session-id>25241</session-id>
</hello>]]>]]>
```

### Hello Message from the Client

```
<?xml version="1.0"?>
<nc:hello xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <nc:capabilities>
    <nc:capability>urn:ietf:params:xml:ns:netconf:base:1.0</nc:capability>
  </nc:capabilities>
</nc:hello>]]>]]>
```

## Obtaining the XSD Files

### Procedure

- Step 1** From your browser, navigate to the Cisco software download site at the following URL:  
<http://software.cisco.com/download/navigator.html>  
The Download Software page opens.
- Step 2** In the Select a Product list, choose **Switches > Data Center Switches > platform > model**.
- Step 3** If you are not already logged in as a registered Cisco user, you are prompted to log in now.
- Step 4** From the Select a Software Type list, choose **NX-OS XML Schema Definition**.

- Step 5** Find the desired release and click **Download**.
- Step 6** If you are requested, follow the instructions to apply for eligibility to download strong encryption software images.  
The Cisco End User License Agreement opens.
- Step 7** Click **Agree** and follow the instructions to download the file to your PC.

## Sending an XML Document to the XML Server

To send an XML document to the XML server through an SSH session that you opened in a command shell, you can copy the XML text from an editor and paste it into the SSH session. Although typically you use an automated method to send XML documents to the XML server, you can verify the SSH connection to the XML server with this method.

Follow these guidelines for this method:

- Verify that the XML server sent the hello message immediately after you started the SSH session by looking for the hello message text in the command shell output.
- Send the client hello message before you send any XML requests. Because the XML server sends the hello response immediately, no additional response is sent after you send the client hello message.
- Always terminate the XML document with the character sequence `]]>]]>`.

## Creating NETCONF XML Instances

You can create NETCONF XML instances by enclosing XML device elements within an RPC tag and NETCONF operation tags. The XML device elements are defined in feature-based XML schema definition (XSD) files, which enclose available CLI commands in an XML format.

The following are the tags that are used in the NETCONF XML request in a framework context. Tag lines are marked with the following letter codes:

- X—XML declaration
- R—RPC request tag
- N—NETCONF operation tags
- D—Device tags

### NETCONF XML Framework Context

```
X <?xml version="1.0"?>
R <nc:rpc message-id="1" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
R xmlns="http://www.cisco.com/nxos:1.0:nfcli">
N <nc:get>
N <nc:filter type="subtree">
D <show>
D <xml>
D <server>
D <status/>
D </server>
D </xml>
D </show>
N </nc:filter>
N </nc:get>
R </nc:rpc>]]>]]>
```



**Note** You must use your own XML editor or XML management interface tool to create XML instances.

## RPC Request Tag `rpc`

All NETCONF XML instances must begin with the RPC request tag `<rpc>`. The example *RPC Request Tag* `<rpc>` shows the `<rpc>` element with its required **message-id** attribute. The message-id attribute is replicated in the `<rpc-reply>` and can be used to correlate requests and replies. The `<rpc>` node also contains the following XML namespace declarations:

- NETCONF namespace declaration—The `<rpc>` and NETCONF tags that are defined in the "urn:ietf:params:xml:ns:netconf:base:1.0" namespace, are present in the `netconf.xsd` schema file.
- Device namespace declaration—Device tags encapsulated by the `<rpc>` and NETCONF tags are defined in other namespaces. Device namespaces are feature-oriented. Cisco NX-OS feature tags are defined in different namespaces. *RPC Request Tag* `<rpc>` is an example that uses the `nfcli` feature. It declares that the device namespace is "xmlns=http://www.cisco.com/nxos:1.0:nfcli". `nfcli.xsd` contains this namespace definition. For more information, see section on *Obtaining the XSD Files*.

### RPC Tag Request

```
<nc:rpc message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
...
</nc:rpc>]]>]]>
```

### Configuration Request

The following is an example of a configuration request.

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
  <nc:edit-config>
    <nc:target>
      <nc:running/>
    </nc:target>
    <nc:config>
      <configure>
        <__XML__MODE__exec_configure>
          <interface>
            <ethernet>
              <interface>2/30</interface>
              <__XML__MODE_if-ethernet>
                <__XML__MODE_if-eth-base>
                  <description>
                    <desc_line>Marketing Network</desc_line>
                  </description>
                </__XML__MODE_if-eth-base>
              </__XML__MODE_if-ethernet>
            </ethernet>
          </interface>
        </__XML__MODE__exec_configure>
      </configure>
    </nc:config>
  </nc:edit-config>
</nc:rpc>]]>]]>
```

\_\_XML\_\_MODE tags are used internally by the NETCONF agent. Some tags are present only as children of a certain \_\_XML\_\_MODE. By examining the schema file, you can find the correct mode tag that leads to the tags representing the CLI command in XML.

## NETCONF Operations Tags

NETCONF provides the following configuration operations:

**Table 16: NETCONF Operations in Cisco NX-OS**

NETCONF Operation	Description	Example
close-session	Closes the current XML server session.	<a href="#">NETCONF Close Session Instance, on page 231</a>
commit	Sets the running configuration to the current contents of the candidate configuration.	<a href="#">NETCONF Commit Instance - Candidate Configuration Capability, on page 236</a>
confirmed-commit	Provides parameters to commit the configuration for a specified time. If this operation is not followed by a commit operation within the confirm-timeout period, the configuration is reverted to the state before the confirmed-commit operation.	<a href="#">NETCONF Confirmed-commit Instance , on page 236</a>
copy-config	Copies the content of source configuration datastore to the target datastore.	<a href="#">NETCONF copy-config Instance, on page 232</a>
delete-config	Operation not supported.	—
edit-config	Configures features in the running configuration of the device. You use this operation for configuration commands.	<a href="#">NETCONF edit-config Instance, on page 232</a> <a href="#">NETCONF rollback-on-error Instance , on page 236</a>
get	Receives configuration information from the device. You use this operation for <b>show</b> commands. The source of the data is the running configuration.	<a href="#">Creating NETCONF XML Instances, on page 221</a>
get-config	Retrieves all or part of a configuration	<a href="#">NETCONF get-config Instance, on page 234</a>
kill-session	Closes the specified XML server session. You cannot close your own session. See the close-session NETCONF operation.	<a href="#">NETCONF Kill-session Instance, on page 232</a>

NETCONF Operation	Description	Example
lock	Allows the client to lock the configuration system of a device.	<a href="#">NETCONF Lock Instance, on page 234</a>
unlock	Releases the configuration lock that the session issued.	<a href="#">NETCONF unlock Instance, on page 235</a>
validate	Checks a candidate configuration for syntactical and semantic errors before applying the configuration to the device.	<a href="#">NETCONF validate Capability Instance , on page 237</a>

## Device Tags

The XML device elements represent the available CLI commands in XML format. The feature-specific schema files contain the XML tags for CLI commands of that particular feature. See the [Obtaining the XSD Files, on page 220](#) section.

Using this schema, it is possible to build an XML instance. In the following examples, the relevant portions of the nfcli.xsd schema file that was used to build [Creating NETCONF XML Instances, on page 221](#) is shown.

The following example shows XML device tags.

### show xml Device Tags

```
<xs:element name="show" type="show_type_Cmd_show_xml"/>
<xs:complexType name="show_type_Cmd_show_xml">
  <xs:annotation>
    <xs:documentation>to display xml agent information</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:choice maxOccurs="1">
      <xs:element name="xml" minOccurs="1" type="xml_type_Cmd_show_xml"/>
      <xs:element name="debug" minOccurs="1" type="debug_type_Cmd_show_debug"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="xpath-filter" type="xs:string"/>
  <xs:attribute name="uses-namespace" type="nxos:bool_true"/>
</xs:complexType>
```

The following example shows the server status device tags.

### server status Device Tags

```
<xs:complexType name="xml_type_Cmd_show_xml">
  <xs:annotation>
    <xs:documentation>xml agent</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="server" minOccurs="1" type="server_type_Cmd_show_xml"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="server_type_Cmd_show_xml">
  <xs:annotation>
    <xs:documentation>xml agent server</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:choice maxOccurs="1">
```



```

<xs:element name="status" minOccurs="1" type="status_type_Cmd_show_xml"/>
<xs:element name="logging" minOccurs="1" type="logging_type_Cmd_show_logging_facility"/>
</xs:choice>
</xs:sequence>
</xs:complexType>

```

The following example shows the device tag response.

### Device Tag Response

```

<xs:complexType name="status_type_Cmd_show_xml">
<xs:annotation>
<xs:documentation>display xml agent information</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="__XML_OPT_Cmd_show_xml__readonly__" minOccurs="0">
<xs:complexType>
<xs:sequence>
<xs:group ref="og_Cmd_show_xml__readonly__" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:group name="og_Cmd_show_xml__readonly__">
<xs:sequence>
<xs:element name="__readonly__" minOccurs="1" type="__readonly__type_Cmd_show_xml"/>
</xs:sequence>
</xs:group>
<xs:complexType name="__readonly__type_Cmd_show_xml">
<xs:sequence>
<xs:group ref="bg_Cmd_show_xml_operational_status" maxOccurs="1"/>
<xs:group ref="bg_Cmd_show_xml_maximum_sessions_configured" maxOccurs="1"/>
<xs:group ref="og_Cmd_show_xml_TABLE_sessions" minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

```



**Note** “\_\_XML\_OPT\_Cmd\_show\_xml\_\_readonly\_\_” is optional. This tag represents the response. For more information on responses, see the [RPC Response Tag, on page 230](#) section.

You can use the | XML option to find the tags you can use to execute a <get>. The following is an example of the | XML option.

### XML Example

```

Switch#> show xml server status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:nfcli">
<nf:data>
<show>
<xml>
<server>
<status>
<__XML_OPT_Cmd_show_xml__readonly__>
<__readonly__>
<operational_status>
<o_status>enabled</o_status>
</operational_status>
<maximum_sessions_configured>

```

```

<max_session>8</max_session>
</maximum_sessions_configured>
</__readonly__>
</__XML_OPT_Cmd_show_xml__readonly__>
</status>
</server>
</xml>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```

From this response, you can see that the namespace defining tag to execute operations on this component is `http://www.cisco.com/nxos:1.0:nfcli` and the `nfcli.xsd` file can be used to build requests for this feature.

You can enclose the NETCONF operation tags and the device tags within the RPC tag. The `</rpc>` end-tag is followed by the XML termination character sequence.

## Extended NETCONF Operations

Cisco NX-OS supports an `<rpc>` operation named `<exec-command>`. The operation allows client applications to send CLI configuration and show commands and to receive responses to those commands as XML tags.

The following is an example of the tags that are used to configure an interface. Tag lines are marked with the following letter codes:

- X—XML declaration
- R—RPC request tag
- EO—Extended operation

### Configuration CLI Commands Sent Through `<exec-command>`

```

X <?xml version="1.0"?>
R <nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
EO <nxos:exec-command>
EO <nxos:cmd>conf t ; interface ethernet 2/1 </nxos:cmd>
EO <nxos:cmd>channel-group 2000 ; no shut; </nxos:cmd>
EO </nxos:exec-command>
R </nf:rpc>]]>]]>

```

The following is the response to the operation:

### Response to CLI Commands Sent Through `<exec-command>`

```

<?xml version="1.0" encoding="ISO-8859-1">
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:ok/>
</nf:rpc-reply>
]]>]]>

```

The following example shows how the show CLI commands that are sent through the `<exec-command>` can be used to retrieve data.

**show CLI Commands Sent Through <exec-command>**

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show interface brief</nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
```

The following is the response to the operation.

**Response to the show CLI commands Sent Through <exec-command>**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0"
xmlns:mod="http://www.cisco.com/nxos:1.0:if_manager" message-id="110">
<nf:data>
<mod:show>
<mod:interface>
<mod:__XML_OPT_Cmd_show_interface_brief__readonly__>
<mod:__readonly__>
<mod:TABLE_interface>
<mod:ROW_interface>
<mod:interface>mgmt0</mod:interface>
<mod:state>up</mod:state>
<mod:ip_addr>172.23.152.20</mod:ip_addr>
<mod:speed>1000</mod:speed>
<mod:mtu>1500</mod:mtu>
</mod:ROW_interface>
<mod:ROW_interface>
<mod:interface>Ethernet2/1</mod:interface>
<mod:vlan>--</mod:vlan>
<mod:type>eth</mod:type>
<mod:portmode>routed</mod:portmode>
<mod:state>down</mod:state>
<mod:state_rsn_desc>Administratively down</mod:state_rsn_desc>
<mod:speed>auto</mod:speed>
<mod:ratemode>D</mod:ratemode>
</mod:ROW_interface>
</mod:TABLE_interface>
</mod:__readonly__>
</mod:__XML_OPT_Cmd_show_interface_brief__readonly__>
</mod:interface>
</mod:show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

The following table provides a detailed explanation of the operation tags:

**Table 17: Tags**

Tag	Description
<exec-command>	Executes a CLI command.

Tag	Description
<cmd>	Contains the CLI command. A command can be a show or configuration command. Separate multiple configuration commands by using a semicolon “;”. Multiple show commands are not supported. You can send multiple configuration commands in different <cmd> tags as part of the same request. For more information, see the Example in <i>Configuration CLI Commands Sent Through &lt;exec-command&gt;</i> .

Replies to configuration commands that are sent through the <cmd> tag are as follows:

- <nf:ok>: All configure commands are executed successfully.
- <nf:rpc-error>: Some commands have failed. The operation stops on the first error, and the <nf:rpc-error> subtree provides more information on what configuration failed. Notice that any configuration that is executed before the failed command would have been applied to the running configuration.

The following example shows a failed configuration:

### Failed Configuration

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nxos:exec-command>
<nxos:cmd>configure terminal ; interface ethernet2/1 </nxos:cmd>
<nxos:cmd>ip address 1.1.1.2/24 </nxos:cmd>
<nxos:cmd>no channel-group 2000 ; no shut; </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="3">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Ethernet2/1: not part of port-channel 2000
</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]]]>
```

Because of a command execution, the interface IP address is set, but the administrative state is not modified (the no shut command is not executed). The reason the administrative state is not modified is because the no port-channel 2000 command results in an error.

The <rpc-reply> results from a show command that is sent through the <cmd> tag that contains the XML output of the show command.

You cannot combine configuration and show commands on the same <exec-command> instance. The following example shows a configuration and **show** command that are combined in the same instance.

## Combination of Configuration and show Commands

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>conf t ; interface ethernet 2/1 ; ip address 1.1.1.4/24 ; show xml
server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: cannot mix config and show in exec-command. Config cmds
before the show were executed.
Cmd:show xml server status</nf:error-message>
<nf:error-info>
<nf:bad-element>cmd</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

The show command must be sent in its own `<exec-command>` instance as shown in the following example:

## Show CLI Commands Sent Through `<exec-command>`

```
<?xml version="1.0"?>
<nf:rpc xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nxos:exec-command>
<nxos:cmd>show xml server status ; show xml server status </nxos:cmd>
</nxos:exec-command>
</nf:rpc>]]>]]>
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nxos="http://www.cisco.com/nxos:1.0" message-id="110">
<nf:rpc-error>
<nf:error-type>application</nf:error-type>
<nf:error-tag>invalid-value</nf:error-tag>
<nf:error-severity>error</nf:error-severity>
<nf:error-message>Error: show cmds in exec-command shouldn't be followed by anything
</nf:error-message>
<nf:error-info>
<nf:bad-element>&lt;cmd&gt;</nf:bad-element>
</nf:error-info>
</nf:rpc-error>
</nf:rpc-reply>
]]>]]>
```

## NETCONF Replies

For every XML request sent by the client, the XML server sends an XML response enclosed in the RPC response tag `<rpc-reply>`.

This section contains the following topics:

- [RPC Response Tag, on page 230](#)
- [Interpreting Tags Encapsulated in the Data Tag, on page 230](#)

## RPC Response Tag

The following example shows the RPC response tag `<rpc-reply>`.

### RPC Response Elements

```
<nc:rpc-reply message-id="315" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns=http://www.cisco.com/nxos:1.0:nfcli">
<ok/>
</nc:rpc-reply>]]]]>
```

The elements `<ok>`, `<data>`, and `<rpc-error>` can appear in the RPC response. The following table describes the RPC response elements that can appear in the `<rpc-reply>` tag.

**Table 18: RPC Response Elements**

Element	Description
<code>&lt;ok&gt;</code>	The RPC request completed successfully. This element is used when no data is returned in the response.
<code>&lt;data&gt;</code>	The RPC request completed successfully. The data associated with the RPC request is enclosed in the <code>&lt;data&gt;</code> element.
<code>&lt;rpc-error&gt;</code>	The RPC request failed. Error information is enclosed in the <code>&lt;rpc-error&gt;</code> element.

## Interpreting Tags Encapsulated in the Data Tag

The device tags encapsulated by the `<data>` tag contain the request followed by the response. A client application can safely ignore all tags before the `<readonly>` tag. The following is an example:

### RPC-reply data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nf:data>
<show>
<interface>
<__XML__OPT_Cmd_show_interface_brief__readonly__>
<__readonly__>
<TABLE_interface>
<ROW_interface>
<interface>mgmt0</interface>
<state>up</state>
<ip_addr>xx.xx.xx.xx</ip_addr>
<speed>1000</speed>
<mtu>1500</mtu>
</ROW_interface>
<ROW_interface>
<interface>Ethernet2/1</interface>
```

```

<vlan>--</vlan>
<type>eth</type>
<portmode>routed</portmode>
<state>down</state>
<state_rsn_desc>Administratively down</state_rsn_desc>
<speed>auto</speed>
<ratemode>D</ratemode>
</ROW_interface>
</TABLE_interface>
</__readonly__>
</__XML__OPT_Cmd_show_interface_brief__readonly__>
</interface>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>

```

<\_\_XML\_\_OPT.\*> and <\_\_XML\_\_BLK.\*> appear in responses and are sometimes used in requests. These tags are used by the NETCONF agent and are present in responses after the <\_\_readonly\_\_> tag. They are necessary in requests and should be added according to the schema file to reach the XML tag that represents the CLI command.

## Information About Example XML Instances

### Example XML Instances

This section provides the examples of the following XML instances:

- [NETCONF Close Session Instance, on page 231](#)
- [NETCONF Kill-session Instance, on page 232](#)
- [NETCONF copy-config Instance, on page 232](#)
- [NETCONF edit-config Instance, on page 232](#)
- [NETCONF get-config Instance, on page 234](#)
- [NETCONF Lock Instance, on page 234](#)
- [NETCONF unlock Instance, on page 235](#)
- [NETCONF Commit Instance - Candidate Configuration Capability, on page 236](#)
- [NETCONF Confirmed-commit Instance, on page 236](#)
- [NETCONF rollback-on-error Instance, on page 236](#)
- [NETCONF validate Capability Instance, on page 237](#)

### NETCONF Close Session Instance

The following example shows the close-session request, followed by the close-session response.

#### Close-session Request

```

<?xml version="1.0"?>
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:close-session/>
</nc:rpc>]]>]]>

```

**Close-session Response**

```
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0" message-id="101">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

**NETCONF Kill-session Instance**

The following example shows the kill-session request followed by the kill-session response.

**Kill-session Request**

```
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:kill-session>
<nc:session-id>25241</nc:session-id>
</nc:kill-session>
</nc:rpc>]]>]]>
```

**Kill-session Request**

```
<nc:rpc message-id="101" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0">
<nc:kill-session>
<nc:session-id>25241</nc:session-id>
</nc:kill-session>
</nc:rpc>]]>]]>
```

**NETCONF copy-config Instance**

The following example shows the copy-config request followed by the copy-config response.

**Copy-config Request**

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy-config>
<target>
<running/>
</target>
<source>
<url>https://user@example.com:passphrase/cfg/new.txt</url>
</source>
</copy-config>
</rpc>
```

**Copy-config Response**

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

**NETCONF edit-config Instance**

The following example shows the use of NETCONF edit-config.



## Edit-config Request

```
<?xml version="1.0"?>
<nc:rpc message-id="16" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager">
<nc:edit-config>
<nc:target>
<nc:running/>
</nc:target>
<nc:config>
<configure>
<_XML_MODE__exec_configure>
<interface>
<ethernet>
<interface>2/30</interface>
<_XML_MODE_if-ethernet>
<_XML_MODE_if-eth-base>
<description>
<desc_line>Marketing Network</desc_line>
</description>
</_XML_MODE_if-eth-base>
</_XML_MODE_if-ethernet>
</ethernet>
</interface>
</_XML_MODE__exec_configure>
</configure>
</nc:config>
</nc:edit-config>
</nc:rpc>]]>]]>
```

## Edit-config Response

```
<?xml version="1.0"?>
<nc:rpc-reply xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos:1.0:if_manager" message-id="16">
<nc:ok/>
</nc:rpc-reply>]]>]]>
```

The operation attribute in edit-config identifies the point in configuration where the specified operation is performed. If the operation attribute is not specified, the configuration is merged into the existing configuration data store. Operation attribute can have the following values:

- create
- merge
- delete

The following example shows how to delete the configuration of interface Ethernet 0/0 from the running configuration.

### Edit-config: Delete Operation Request

```
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
<running/>
</target>
<default-operation>none</default-operation>
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
<top xmlns="http://example.com/schema/1.2/config">
```

```

<interface xc:operation="delete">
<name>Ethernet0/0</name>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>

```

### Response to edit-config: Delete Operation

```

<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>

```

## NETCONF get-config Instance

The following example shows the use of NETCONF get-config.

### Get-config Request to Retrieve the Entire Subtree

```

<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<get-config>
<source>
<running/>
</source>
<filter type="subtree">
<top xmlns="http://example.com/schema/1.2/config">
<users/>
</top>
</filter>
</get-config>
</rpc>]]>]]>

```

### Get-config Response with Results of the Query

```

<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<data>
<top xmlns="http://example.com/schema/1.2/config">
<users>
<user>
<name>root</name>
<type>superuser</type>
<full-name>Charlie Root</full-name>
<company-info>
<dept>1</dept>
<id>1</id>
</company-info>
</user>
<!-- additional <user> elements appear here... -->
</users>
</top>
</data>
</rpc-reply>]]>]]>

```

## NETCONF Lock Instance

The following example shows the use of NETCONF lock operation.

The following examples show the lock request, a success response, and a response to an unsuccessful attempt.

### Lock Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<lock>
<target>
<running/>
</target>
</lock>
</rpc>]]>]]>
```

### Response to Successful Acquisition of Lock

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/> <!-- lock succeeded -->
</rpc-reply>]]>]]>
```

### Response to Unsuccessful Attempt to Acquire the Lock

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rpc-error> <!-- lock failed -->
<error-type>protocol</error-type>
<error-tag>lock-denied</error-tag>
<error-severity>error</error-severity>
<error-message>
Lock failed, lock is already held
</error-message>
<error-info>
<session-id>454</session-id>
<!-- lock is held by NETCONF session 454 -->
</error-info>
</rpc-error>
</rpc-reply>]]>]]>
```

## NETCONF unlock Instance

The following example shows the use of the NETCONF unlock operation.

### unlock request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<unlock>
<target>
<running/>
</target>
</unlock>
</rpc>
```

### response to unlock request

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
<ok/>
</rpc-reply>
```

## NETCONF Commit Instance - Candidate Configuration Capability

The following example shows the commit operation and the commit reply:

### Commit Operation

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit/>
</rpc>
```

### Commit Reply

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>
```

## NETCONF Confirmed-commit Instance

The following example shows the confirmed-commit operation and the confirmed-commit reply.

### Confirmed Commit Request

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<commit>
<confirmed/>
<confirm-timeout>120</confirm-timeout>
</commit>
</rpc>]]>]]>
```

### Confirmed Commit Response

```
<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>
```

## NETCONF rollback-on-error Instance

The following example shows the use of NETCONF rollback on error capability. The string `urn:ietf:params:netconf:capability:rollback-on-error:1.0` identifies the capability.

The following example shows how to configure rollback on error and the response to this request.

### Rollback-on-error capability

```
<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<edit-config>
<target>
<running/>
```

```

</target>
<error-option>rollback-on-error</error-option>
<config>
<top xmlns="http://example.com/schema/1.2/config">
<interface>
<name>Ethernet0/0</name>
<mtu>100000</mtu>
</interface>
</top>
</config>
</edit-config>
</rpc>]]>]]>

```

### Rollback-on-error response

```

<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>

```

## NETCONF validate Capability Instance

The following example shows the use of the NETCONF validate capability. The string `urn:ietf:params:netconf:capability:validate:1.0` identifies the capability.

### Validate request

```

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<validate>
<source>
<candidate/>
</source>
</validate>
</rpc>]]>]]>

```

### Response to validate request

```

<rpc-reply message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<ok/>
</rpc-reply>]]>]]>

```

## Additional References

This section provides additional information that is related to implementing the XML management interface.

### Standards

Standards	Title
No new or modified standards are supported by this feature. Support for existing standards has not been modified by this feature.	—

**RFCs**

<b>RFCs</b>	<b>Title</b>
<a href="#">RFC 4741</a>	NETCONF Configuration Protocol
<a href="#">RFC 4742</a>	Using the NETCONF Configuration Protocol over Secure Shell (SSH)