AUtomation and Scripting tools

Introduction to SD-Access

Network automation can make your life easier. There are many reasons why IT organizations of all shapes and sizes are adopting network automation. Following are some of them:

- **Reduced operational costs:** Automation can give return on investment for every dollar spent. Automation has the potential to reduce IT department staffing costs while potentially redirecting IT staff efforts to more proactive, business value–generating projects.

- **Deterministic outcomes:** Automation of the network decreases manual, error-prone processes on network configuration and deployment.

- **Resilient networks:** Automation can detect and fix network errors on the fly without manual intervention. This results in more resilient networks.

- **Faster deployment:** Network automation can deploy and upgrade network devices in the network faster without requiring any manual intervention or configuration.

EEM Overview [Embededded Event Manager]

This section describes how to configure the Embedded Event Manager (EEM) to detect and handle critical events on Cisco NX-OS devices.

The EEM monitors events that occur on the device and takes action to recover or

troubleshoot these events based on the configuration.
The EEM consists of three major components:

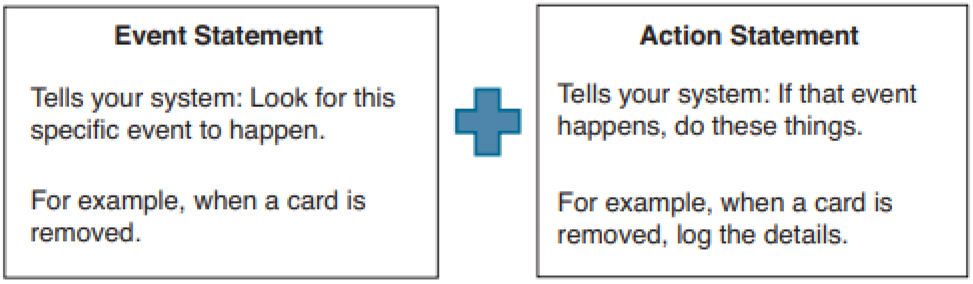
■ **Event statements:** Events to monitor from another Cisco NX-OS component that may require some action, workaround, or notification.

■ **Action statements:** Actions that the EEM can take, such as sending an email or disabling an interface, to recover from an event.

■ **Policies:** Events paired with one or more actions to troubleshoot or recover from the event.

Policies

An EEM policy consists of an event statement and one or more action statements. The event statement defines the event to look for as well as the filtering characteristics for the event.
The action statement defines the action EEM takes when the event occurs.



Event Statement

Tells your system: Look for this specific event to happen.

For example, when a card is removed.

Action Statement

Tells your system: If that event happens, do these things.

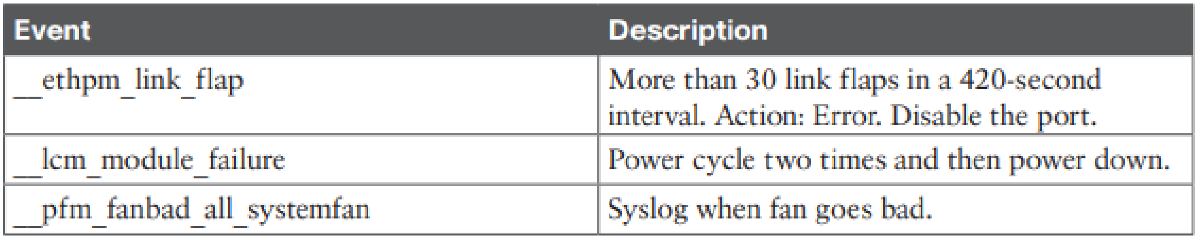
For example, when a card is removed, log the details.

EEM policies can be configured using the **command-line interface (CLI) or a VSH script.**

The EEM gives a device-wide view of policy management. EEM policies are configured on the supervisor, and the EEM pushes the policy to the correct module based on the event type. The EEM takes any actions for a triggered event either locally on the module or on the supervisor (the default option). The EEM maintains event logs on the supervisor.

Cisco NX-OS has a number of preconfigured system policies. These system policies define many common events and actions for the device. **System policy names begin with two underscore characters (__)**. You can create user policies to suit your network. If you create a user policy, any actions in your policy occur after EEM triggers any system policy actions related to the same event as your policy. **You can also override some system policies. The overrides that you configure take the place of the system policy. You can override the event or the actions.**

Use the show event manager system-policy command to view the preconfigured system policies and determine which policies that you can override.



Event	Description
__ethpm_link_flap	More than 30 link flaps in a 420-second interval. Action: Error. Disable the port.
__lcm_module_failure	Power cycle two times and then power down.
__pfm_fanbad_all_systemfan	Syslog when fan goes bad.

Event Statements

An event is any device activity for which some action, such as a workaround or a notification, should be taken. In many cases, these events are related to faults in the device such as when an interface or a fan malfunctions.

The EEM defines event filters so only critical events or multiple occurrences of an event within a specified time period trigger an associated action. Event statements specify the event that triggers a policy to run .

Action Statements

Action statements describe the action triggered by a policy. Each policy can have multiple
action statements. If no action is associated with a policy, EEM still observes events but
takes no actions

EEM supports the following actions in action statements:

- **Execute any CLI commands.**
- **Update a counter.**
- **Log an exception.**
- **Force the shutdown of any module.**
- **Reload the device.**
- **Shut down specified modules because the power is over budget.**
- **Generate a syslog message.**
- **Generate a Call Home event.**
- **Generate an SNMP notification.**
- **Use the default action for the system policy.**

If you want to allow the triggered event to process any default actions, you must configure
the EEM policy to allow the default action. For example, if you match a CLI command in a
match statement, you must add the event-default action statement to the EEM policy; otherwise, EEM will not allow the CLI command to execute.

important note: Verify that your action statements within your user policy or overriding policy do not negate each other or adversely affect the associated system policy

NOTE: The username: admin (with network-admin or vdc-admin user privileges) is required
to configure EEM on a nondefault VDC.

Configuring EEM

EEM configuration is a three-step process:

Step 1. Register the applet with the EEM and enter applet configuration mode.

```
event manager applet applet-name
```

Step 2. Configure the event statement for the policy. Repeat this step for multiple event
statements.

```
event event-statement
```

Step 3. Configure an action statement for the policy. Repeat this step for multiple
action statements.

```
action number[.number2] action-statement
```

You can define environment variables for EEM that are available for all policies. **Environment
variables are useful for configuring common values that you can use in multiple policies**
.
For example, you can create an environment variable for the IP address of an external email
server:

```
event manager environment variable-name variable-value
```

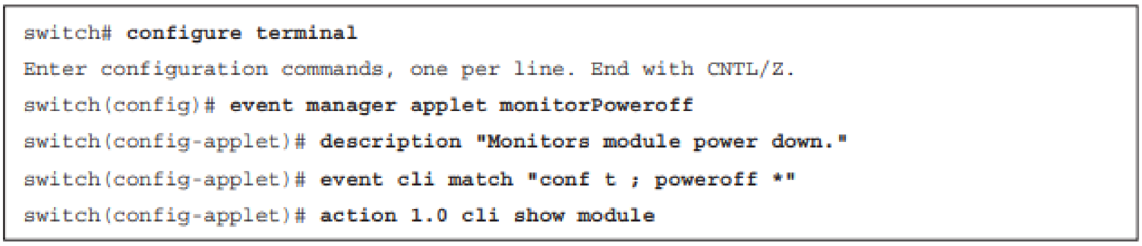
The variable-name can be **any case-sensitive alphanumeric string up to 29 characters**. The
variable-value can be
any quoted alphanumeric string up to 39 characters.

You can also override a system policy using the following command:

```
event manager applet applet-name override system-policy
```

The applet-name can be **any case-sensitive alphanumeric string up to 29 characters**. The

system-policy must be one of the existing system policies.



```
switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
switch(config)# event manager applet monitorPoweroff
switch(config-applet)# description "Monitors module power down."
switch(config-applet)# event cli match "conf t ; poweroff *"
switch(config-applet)# action 1.0 cli show module
```

Verifying the EEM Configuration

You can use the following commands to verify the EEM configuration:

show running-config eem: Displays information about the running configuration for EEM.

show event manager system-policy [all]: Displays information about the predefined system policies.

show event manager policy active detailed: Displays the EEM policies that are executing.

Scheduler

In regular day-to-day network operations, you need to perform multiple routine maintenance activities on a regular basis —for example, backing up data or saving a configuration. **You can achieve these goals by using the Scheduler**. **The Scheduler uses jobs that consist of a**

single command or multiple commands that define routine activities
. **Jobs can be scheduled**
one time or at periodic intervals
.

The Scheduler defines a job and its timetable as follows:

- ■ **Job:** A routine task defined as a command list and completed according to a specified schedule.

- ■ **Schedule:** The timetable for completing a job. You can assign multiple jobs to a schedule. A schedule is defined as either periodic or one-time only.

- ■ **Periodic mode:** A recurring interval that continues until you delete the job. You can configure the following types of intervals:

- ■ **Daily:** A job is completed once a day.

- ■ **Weekly:** A job is completed once a week.

- ■ **Monthly:** A job is completed once a month.

- ■ **Delta:** A job begins at the specified start time and then at specified intervals (days:hours:minutes).

- ■ **One-time mode:** A job is completed only once at a specified time.

Before starting a job, the **Scheduler authenticates the user who created the job**. **Because user credentials from a remote authentication are not retained long enough to support a scheduled job , you need to locally configure the authentication passwords for users who create jobs.**
These passwords are part of the Scheduler configuration and are not considered

a locally configured user. Before starting the job, the Scheduler validates the local password against the password from the remote authentication server.

NOTE The Scheduler requires no license

The Scheduler has the following prerequisites:

- You must enable any conditional features before you can configure those features in a job.
- You must have a valid license installed for any licensed features that you want to configure in the job.
- You must have network-admin user privileges to configure a scheduled job.

Configuring Scheduler

The Scheduler has the following configuration guidelines and limitations:

- The Scheduler can fail if it encounters one of the following while performing a job:
- If the license has expired for a feature at the time the job for that feature is scheduled.
- If a feature is disabled at the time when a job for that feature is scheduled.
- If you have removed a module from a slot and a job for that slot is scheduled.
- Verify that you have configured the time. The Scheduler does not apply a default timetable. If you create a schedule and assign jobs and do not configure the time, the job is not started.
- While defining a job, verify that no interactive or disruptive commands (for example, copy bootflash: file ftp: URI, write erase, and other similar commands) are specified because the job is started and conducted noninteractively.

You can enable the Scheduler feature so that you can configure and schedule jobs, or you can disable the Scheduler feature after it has been enabled:

```
switch(config)# [no] feature scheduler
```

You can configure the log file size for capturing jobs, schedules, and job output:

```
switch(config)# scheduler logfile size value
```

In this command, value defines the scheduler log file size in kilobytes

NOTE If the size of the job output is greater than the size of the log file, the output is
truncated.

You can configure the scheduler to use remote authentication for users who want to configure and schedule jobs. The following command **configures a cleartext password for the user who is currently logged in**:

```
switch(config)# scheduler aaa-authentication password [0 | 7]
```

The following command **configures a cleartext password for a remote user**:

```
switch(config)# scheduler aaa-authentication username name
password [0 | 7] password
```

NOTE Remote users must authenticate with their cleartext password before creating and
configuring jobs.

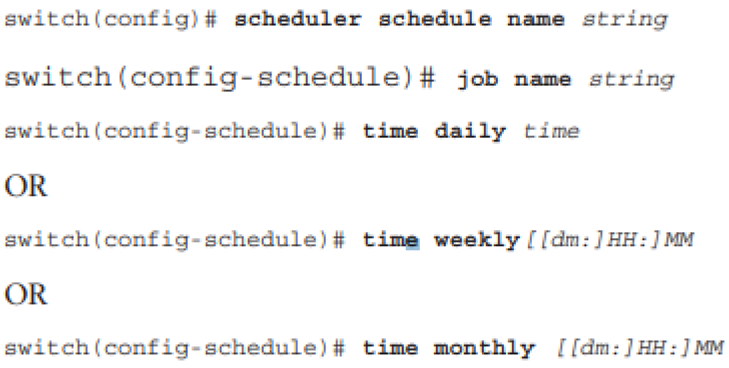
You can define a job including the job name and the command sequence. The following two
commands create a job an d define the sequence of commands for the specified job:

```
switch(config)# scheduler job name string
switch(config-job)# command1 ;[command2;command3 ;...]
```

You can define a timetable in the Scheduler to be used with one or more jobs. If you do
not specify the time for the time commands, the Scheduler assumes the current time. You
can start a job daily at a designated time specified as HH:MM. You can also start a job on a

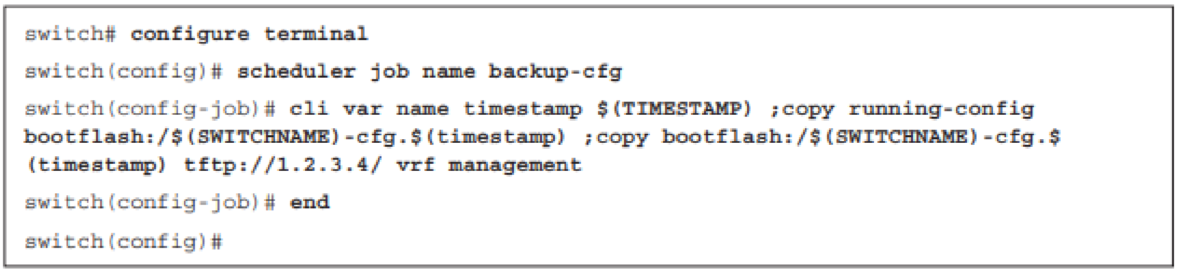
specified day of the week specified as follows:

- **An integer such as 1 = Sunday, 2 = Monday, and so on.**
- **An abbreviation such as Sun = Sunday**



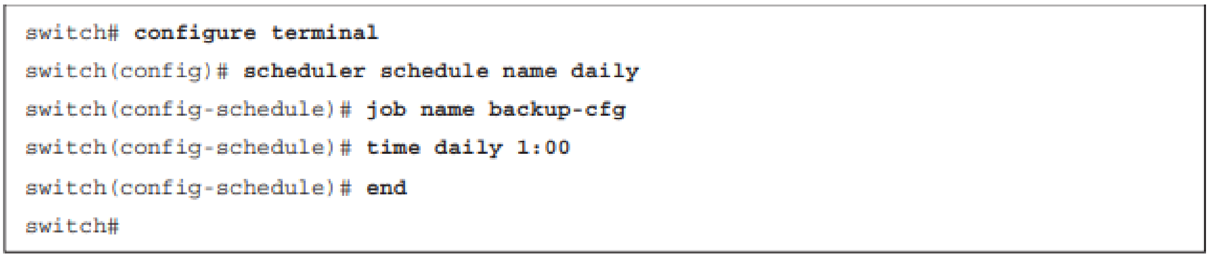
```
switch(config)# scheduler schedule name string

switch(config-schedule)# job name string

switch(config-schedule)# time daily time

OR

switch(config-schedule)# time weekly [[dm:]HH:]MM

OR

switch(config-schedule)# time monthly  [[dm:]HH:]MM
```

Example shows how to create a Scheduler job that saves the running configuration to
a file in bootflash and then copies the file from bootflash to a TFTP server (the filename is
created using the current timestamp and switch name).



```
switch# configure terminal
switch(config)# scheduler job name backup-cfg
switch(config-job)# cli var name timestamp $(TIMESTAMP) ;copy running-config
bootflash:/$(SWITCHNAME)-cfg.$(timestamp) ;copy bootflash:/$(SWITCHNAME)-cfg.$
(timestamp) tftp://1.2.3.4/ vrf management
switch(config-job)# end
switch(config)#
```

Example shows how to schedule a Scheduler job called backup-cfg to run daily at
1 a.m.



```
switch# configure terminal
switch(config)# scheduler schedule name daily
switch(config-schedule)# job name backup-cfg
switch(config-schedule)# time daily 1:00
switch(config-schedule)# end
switch#
```

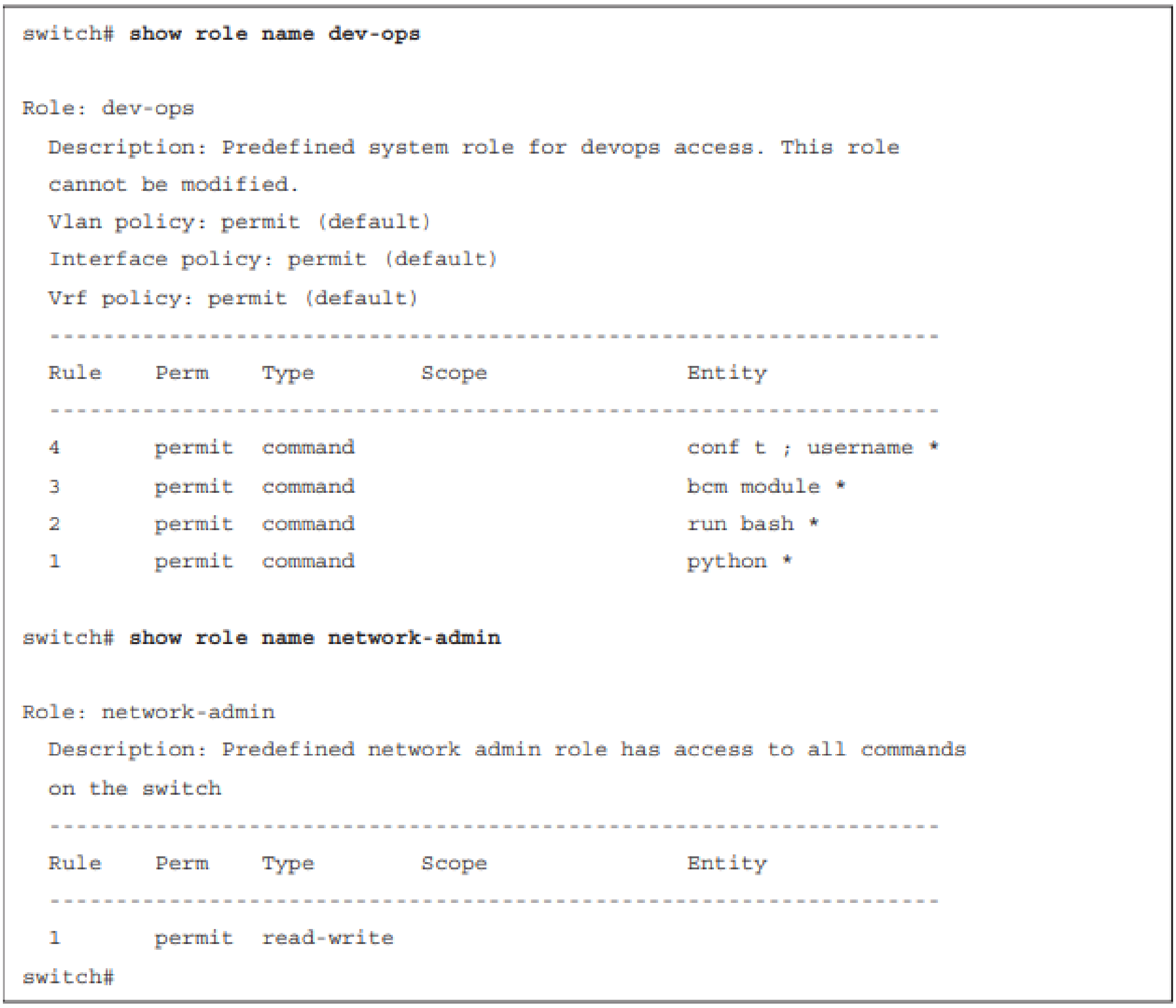
Bash Shell for Cisco NX-OS

In addition to the NX-OS CLI, Cisco Nexus 9000, 3000, 3500, 3600, 7000, and 7700 Series switches 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.

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.

Example shows the authority of the dev-ops role and the network-admin role.

Example 15-5 *Displaying Authority of the dev-ops and network-admin Roles*



```
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#
```

You can enable Bash by running the feature bash-shell command. The run bash command
loads Bash and begins at the home directory for the user.

very Important : You can also run Bash by configuring the user shelltype:

```
username foo shelltype bash
```

This command puts you directly into the Bash shell upon login. This does not require
feature bash-shell to be enabled. You can also run NX-OS CLI commands from the Bash
shell using the

vsh -c command

Managing Feature RPMs

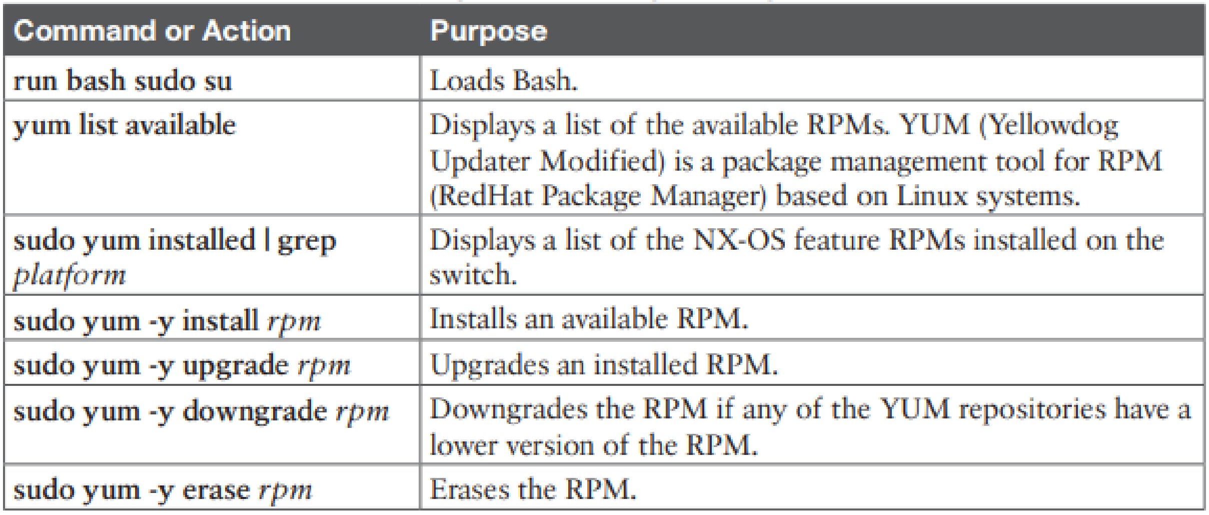
Features on the Nexus 9000, 3000, and 3500 Series are distributed as packages. You can use
the Bash shell to manage those packages. Before installing the RPM package, you need to
verify the system readiness for the same by using the following command:

```
switch# show logging logfile | grep -i "System ready"
```

If you see "System ready" output, you are all set.

Table 15-5 provides some of the commands that you can use to manage RPM packages using
Bash.

Table 15-5 Commands to Manage RPM Packages Using Bash

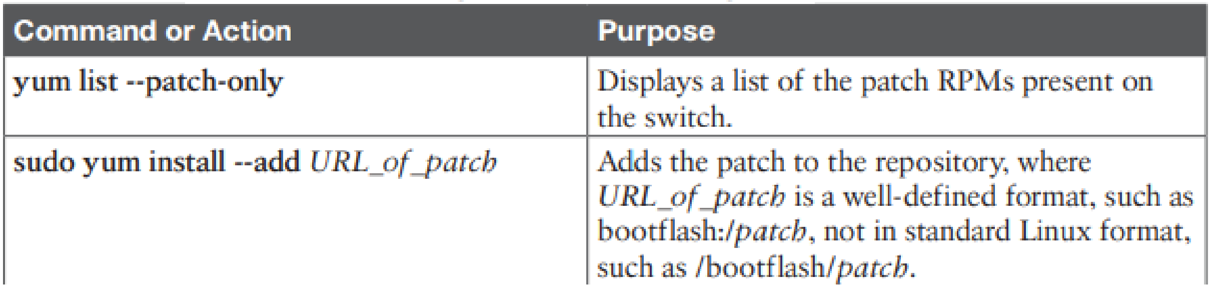


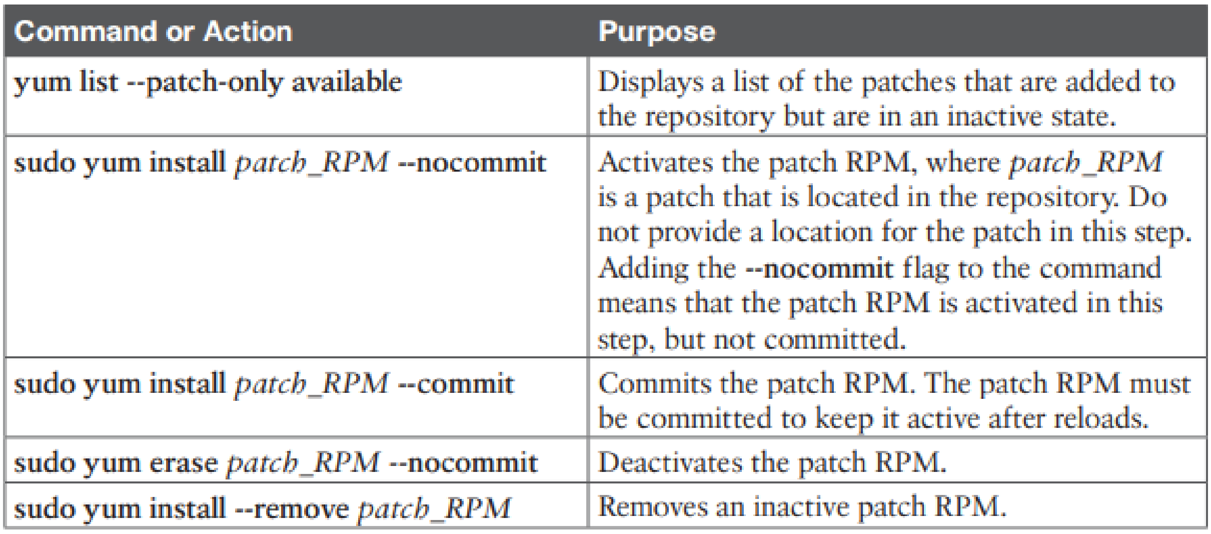
Command or Action	Purpose
run bash sudo su	Loads Bash.
yum list available	Displays a list of the available RPMs. YUM (Yellowdog Updater Modified) is a package management tool for RPM (RedHat Package Manager) based on Linux systems.
sudo yum installed | grep *platform*	Displays a list of the NX-OS feature RPMs installed on the switch.
sudo yum -y install *rpm*	Installs an available RPM.
sudo yum -y upgrade *rpm*	Upgrades an installed RPM.
sudo yum -y downgrade *rpm*	Downgrades the RPM if any of the YUM repositories have a lower version of the RPM.
sudo yum -y erase *rpm*	Erases the RPM.

Managing Patch RPMs

Table 15-6 shows some of the commands that you can use to manage Patch RPMs using
Bash.

Table 15-6 Commands to Manage Patch RPMs Using Bash





Command or Action	Purpose
yum list --patch-only	Displays a list of the patch RPMs present on the switch.
sudo yum install --add *URL_of_patch*	Adds the patch to the repository, where *URL_of_patch* is a well-defined format, such as bootflash:/*patch*, not in standard Linux format, such as /bootflash/*patch*.
yum list --patch-only available	Displays a list of the patches that are added to the repository but are in an inactive state.
sudo yum install *patch_RPM* **--nocommit**	Activates the patch RPM, where *patch_RPM* is a patch that is located in the repository. Do not provide a location for the patch in this step. Adding the **--nocommit** flag to the command means that the patch RPM is activated in this step, but not committed.
sudo yum install *patch_RPM* **--commit**	Commits the patch RPM. The patch RPM must be committed to keep it active after reloads.
sudo yum erase *patch_RPM* **--nocommit**	Deactivates the patch RPM.
sudo yum install --remove *patch_RPM*	Removes an inactive patch RPM.

Guest Shell for Cisco NX-OS

In addition to the NX-OS CLI and Bash access on the underlying Linux environment, the
Cisco Nexus 9000 Series devices support access to a decoupled execution space running
within a Linux Container (LXC) called the
Guest Shell. **When running in the Guest Shell, you have network-admin privileges.**

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

- Access to the network over Linux network interfaces
- Access to Cisco Nexus switch bootflash
- Access to Cisco Nexus switch volatile tmpfs
- Access to Cisco Nexus switch 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.**

Accessing the Guest Shell

You can 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 that is 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.

The Cisco NX-OS automatically installs and enables the Guest Shell by default on systems
with sufficient resources. Subsequent upgrades to the Cisco Nexus series switch software
will not automatically upgrade the Guest Shell. The Guest Shell is based on a

CentOS 7 root file system.

NOTE: Systems with 4 GB of RAM will not enable the Guest Shell by default. The Guest Shell is automatically enabled on systems with more than 4 GB of RAM.

The Guest Shell starts an OpenSSH server upon bootup. 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 virtual-shell when the guestshell keyword is entered. If this server is killed or its configuration (residing in /etc/ssh/sshd_config-cisco) is altered, access to the Guest Shell from the NX-OS CLI might not work.

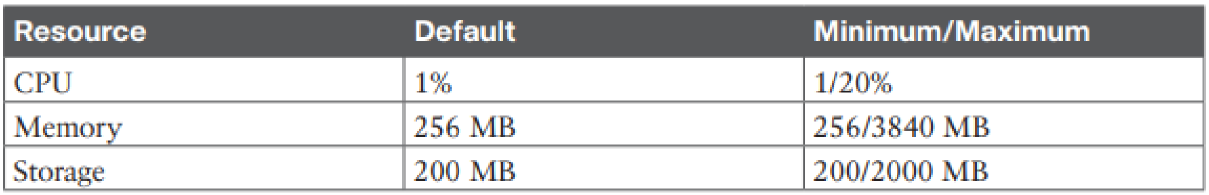
Starting in 2.2(0.2), the Guest Shell will dynamically create user accounts with the same username with which the user logged in to the switch. **However, all other information is NOT shared between the switch and the Guest Shell user accounts.**

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

Resources Used for the Guest Shell

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

Table 15-7 Guest Shell Resource Limits



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

The CPU limit is the percentage of the system compute capacity that tasks running within
the Guest Shell are given when there is contention with other compute loads in the system.

When there is no contention for CPU resources, the tasks within the Guest Shell are not
limited.

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

Misbehaving or malicious application code **can cause DoS as the result of overconsumption**
of connection bandwidth
, disk space, memory, and other resources. The host provides
resource-management features that ensure fair allocation of resources between the Guest
Shell and services on the host.

Capabilities in the Guest Shell

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 prepopulated 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.

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, as shown in Example 15-7. When an interface is placed into
a Virtual Routing and Forwarding (VRF) in the NX-OS CLI, the Linux network interface is

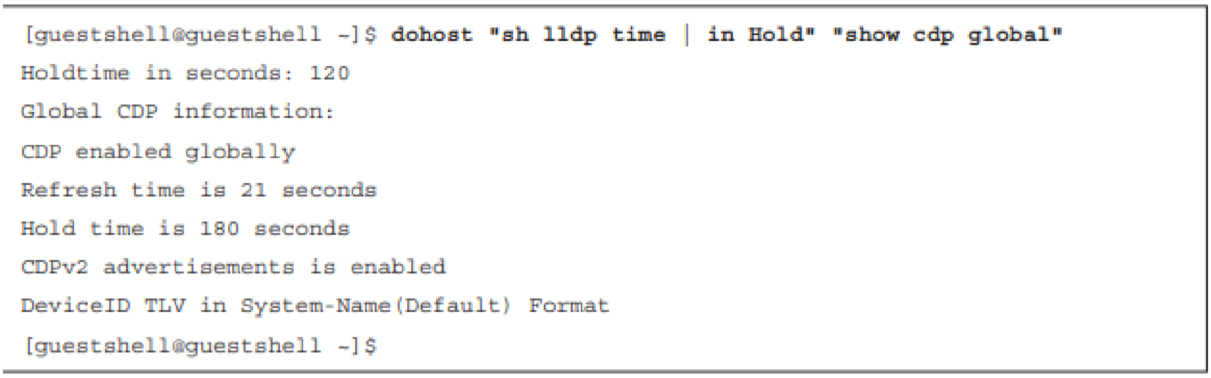
placed into a network namespace for that VRF. You can see the name spaces at /var/run/netns and can use the **ip netns** utility to run in the context of different namespaces. A couple of utilities, **chvrf** and **vrfinfo**, are provided as a convenience for running in a different namespace and getting information about which **namespace or VRF a process is running in**.

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 you are invoking the dohost command, each NX-OS command may be in single or double quotes:

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

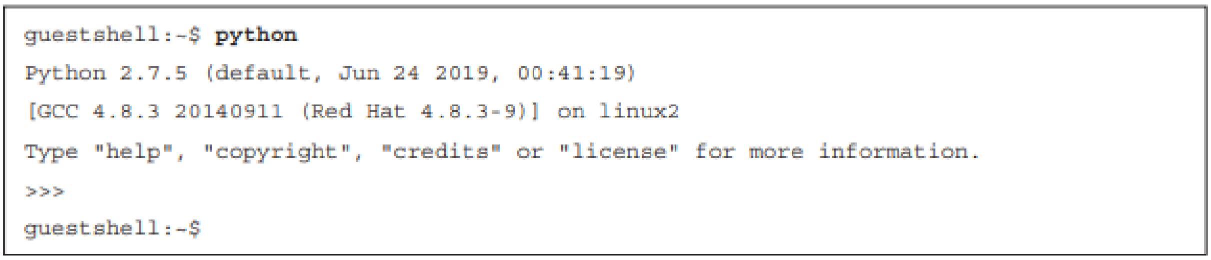
Example 15-8 *Using the* dohost *Command*



```
[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 ~]$
```

Python can be used interactively, or Python scripts can be run in the Guest Shell, as shown in Example

Example 15-9 *Running Python Inside the Guest Shell*



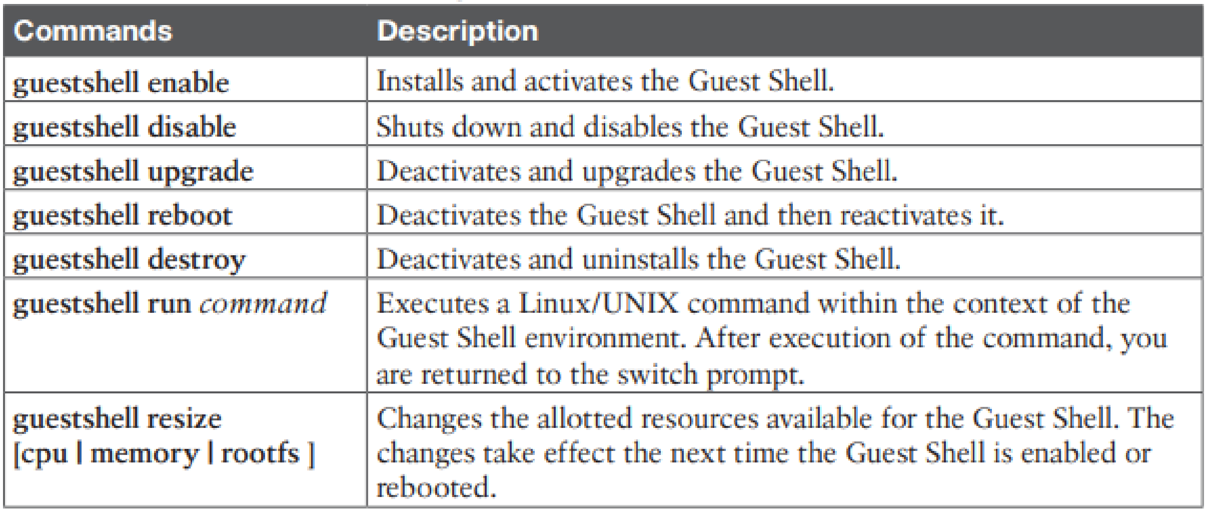
```
guestshell:~$ python
Python 2.7.5 (default, Jun 24 2019, 00:41:19)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
guestshell:~$
```

NOTE: 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. For the host, bootflash: and volatile: 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

lists commands to manage the Guest Shell.

Table 15-8 Commands to Manage the Guest Shell



Commands	Description
guestshell enable	Installs and activates the Guest Shell.
guestshell disable	Shuts down and disables the Guest Shell.
guestshell upgrade	Deactivates and upgrades the Guest Shell.
guestshell reboot	Deactivates the Guest Shell and then reactivates it.
guestshell destroy	Deactivates and uninstalls the Guest Shell.
guestshell run *command*	Executes a Linux/UNIX command within the context of the Guest Shell environment. After execution of the command, you are returned to the switch prompt.
guestshell resize [cpu | memory | rootfs]	Changes the allotted resources available for the Guest Shell. The changes take effect the next time the Guest Shell is enabled or rebooted.

XML

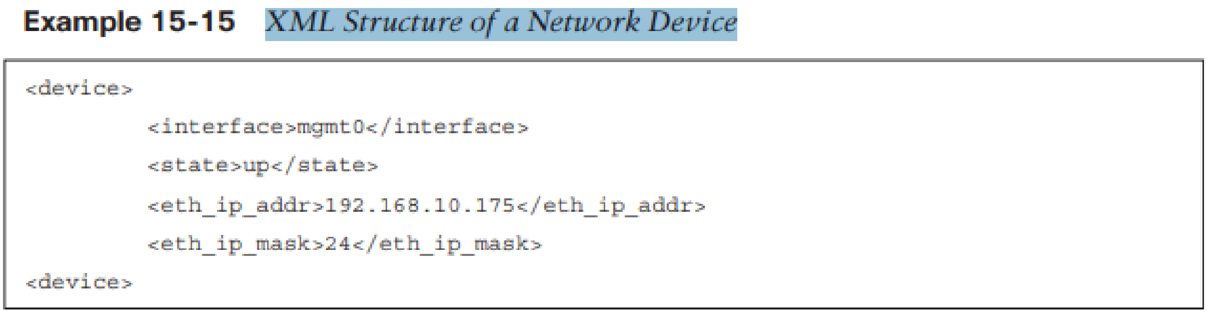
Extensible Markup Language (XML) is a markup language that defines a **set of rules for encoding documents in a format that is both human-readable and machine-readabl**

e. XML
follows a specific format and helps give structure to data. Because XML is **platform-neutral**,

computer-language-neutral, and **text-based,** it is **useful for data exchange between computers and for data storage**

.

the example bellow displays a fragment from an XML document that shows how you might
structure some simple data about a network device.



Example 15-15 *XML Structure of a Network Device*

```
<device>
        <interface>mgmt0</interface>
        <state>up</state>
        <eth_ip_addr>192.168.10.175</eth_ip_addr>
        <eth_ip_mask>24</eth_ip_mask>
<device>
```

The XML fragment has a **root element called device**. The device element has **four child elements**

:

- **interface**
- **state**
- **eth_ip_addr**
- **eth_ip_mask**

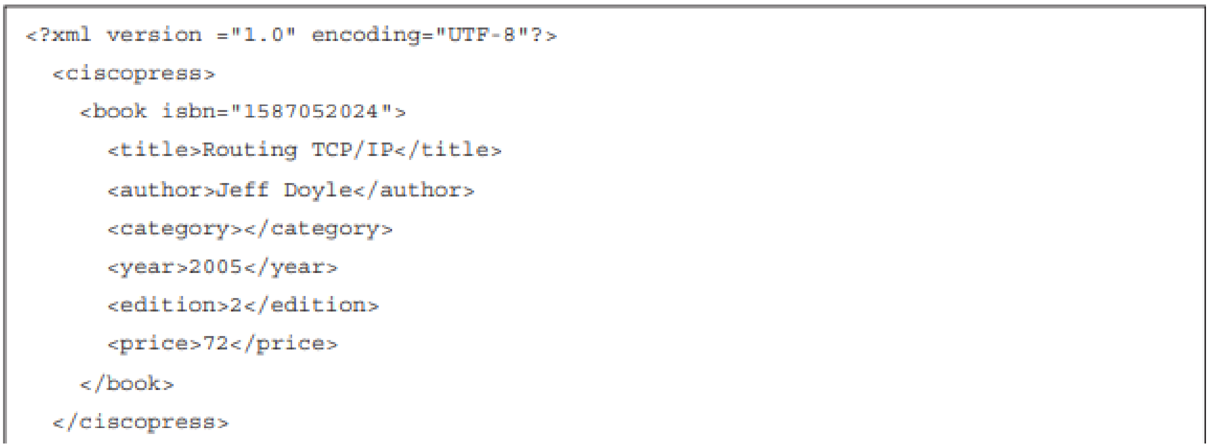
You can think of each element as a data field. XML provides structure to those data fields.
XML doesn’t do anything with the data. To manipulate that data, a piece of software has to

send, receive, store, or display it. **One example of such software is Google Postman**.

6 displays an XML document that has a root node element called ciscopress. Notice that the ciscopress element contains one child element:

book.

Example 15-16 *XML Structure Example*



```
<?xml version ="1.0" encoding="UTF-8"?>
  <ciscopress>
    <book isbn="1587052024">
      <title>Routing TCP/IP</title>
      <author>Jeff Doyle</author>
      <category></category>
      <year>2005</year>
      <edition>2</edition>
      <price>72</price>
    </book>
  </ciscopress>
```

the example below shows a visual representation of the sample XML document as a tree.

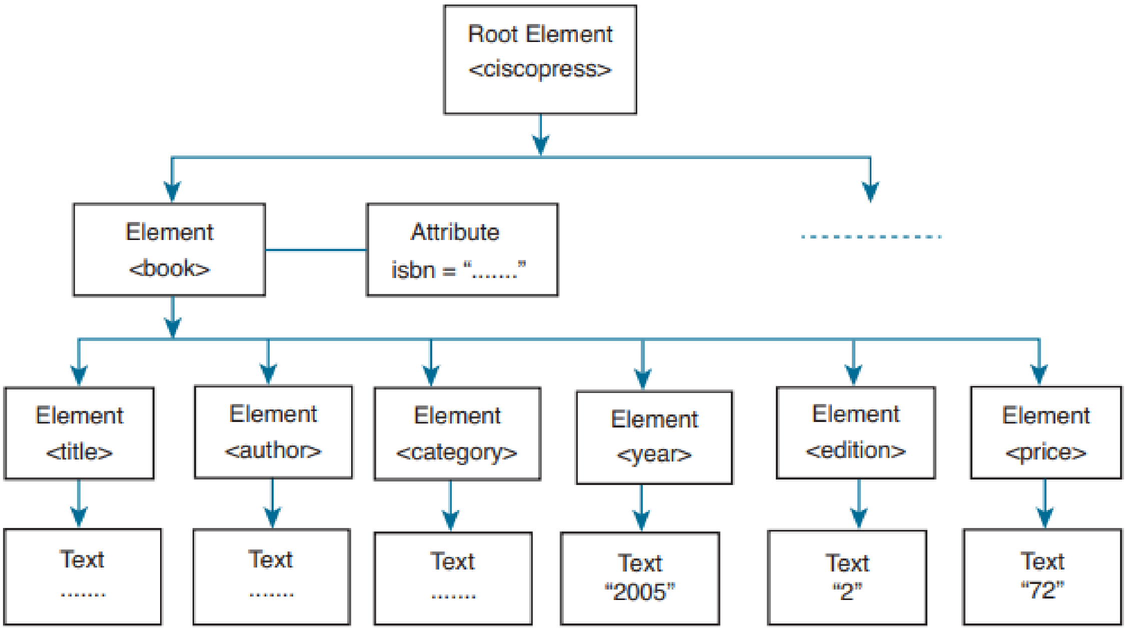


Figure 15-2 *Visual Representation of XML Document as a Tree*

Note the following points:

- The book element contains multiple child elements.
- The book element contains metadata in the form of an attribute called isbn.

An element can contain multiple attributes.

- The <category> element in the sample XML document contains no character

data.
It’s an “empty element.” This is valid if the underlying schema allows it.

One aspect of the XML language that sets it apart from a markup language like HTML is
that it has no predefined tags. The elements in Figure 15-2, such as and ,
are not defined in the XML standard. These elements and the document structure are
created by the author of the XML document. That’s different from the way HTML works
with predefined tags such as <p>, <h1>
, and <table>.

An XML document that contains data is also sometimes referred to as an XML payload.

XML Syntax

The basic unit of an XML document is called an **element**. **Each XML element must conform
to the following rules to be considered “well formed”**
:

Each XML element must have a start-tag and an end-tag. The tags are enclosed in
angle brackets—for example, <device>...</device>. Unlike HTML, the closing tag is
mandatory.
■ An empty-element tag (or standalone tag) must be properly closed—for example,
<interface></interface>.

■ An XML element includes its start-tag, enclosing character data and/or child elements,
and the end-tag.
■ The element’s name can contain letters, numbers, and other Unicode characters, but
NOT white spaces.
■ The element’s name must start with a letter, an underscore “_”, or a colon “:”,
but cannot start with certain reserved words such as xml.
■ Each XML document must have one (and only one) root element. XML elements

must be properly nested. For example,<device><interface>... </device> </interface is incorrectly nested. The correct nesting is <device><interface>...</interface> </device>.

- XML is case sensitive. For example,<Interface> and <interface> are considered two different elements.
- The start-tag may contain attributes in the form of attribute_name="attribute_value". Attributes are used to provide extra information about the element. Unlike HTML, the attribute_value of an XML attribute must be properly quoted (either in double quotes or single quotes).
- Certain characters, such as <, >, which are used in XML syntax, must be replaced with entity references in the form of &entity_name;. XML has five predefined entity references:
- < (<)
- > (>)
- & (&)
- " (")
- ' (').
- XML comments can be used in the form <!-- comment texts -->, which is the same as HTML.
- Unlike HTML, white spaces in the text are preserved.
- A new-line character is represented by a Line Feed (LF) character (0AH).

JSON

JavaScript Object Notation (JSON) is a lightweight text-based open standard designed for human-readable data interchange.

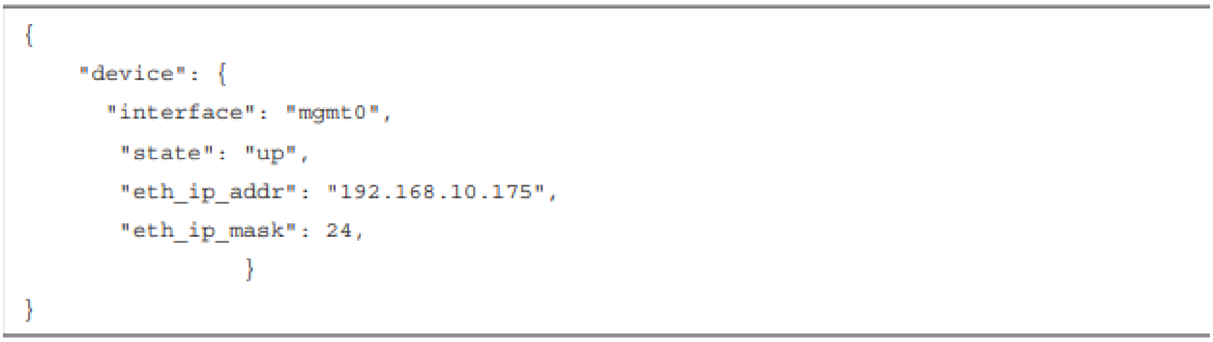
JSON objects contain data in a consistent format that can be passed and programmatically consumed more easily than the data in report formats. A JSON object is an unordered set of name/value pairs, so it tends to be self-explanatory, like XML, but it is less bulky

Parameters for JSON objects are passed in the following format:

ParameterName:parameterValue. A proper JSON object begins with a **left brace { and ends with a right brace }.**
Each name in a pair is followed by a **colon (:)** and then the corresponding value.
The name/value pairs are separated by commas.

the example below displays a fragment from a JSON document that shows how you might structure some simple data about a network device.

Example 15-17 *JSON Structure of a Network Device*



```
{
    "device": {
      "interface": "mgmt0",
       "state": "up",
       "eth_ip_addr": "192.168.10.175",
       "eth_ip_mask": 24,
                }
}
```

Both JSON and XML are human-readable formats. They are both independent of any specific programming language. However, there are a few differences:

■ **Verbose**: XML is more verbose than JSON and usually uses more characters to express the same data than JSON.

■ **Arrays**: XML is used to describe structured data, which doesn't include arrays, whereas JSON includes arrays.

■ **Parsing**: Although most programming languages contain libraries that can parse both JSON and XML data, evaluating specific data elements can be more difficult in

XML
(although more powerful) compared to JSON.

Rest API

An application programming interface (API) is a way for two pieces of software to talk to
each other. An API allows for the development of rich applications with a wide variety of
functionality. Let's go through an example.
Suppose you are the creator of an online marketplace named Jack's Shop, where people can
come and buy stuff and get it delivered to their home/office. How do you track which user
purchased what? You need to maintain a database with user accounts and user order history.
But you don't want to maintain a user credentials database in-house. You would like your
users to log in using their Google or Facebook accounts. How do you achieve this? A simple
answer to this question would be using the Facebook API or Google API to authenticate
users.
REST is centered around the HTTP request and response model. Consuming an API is just
as simple as making an HTTP request. For example, if you make a request to an API Service,
the result of the request will be returned in the response. The data returned in the response
is usually JSON or XML.
To construct a request, you need to know the following information for the API that you are
calling. You can find this information in the API reference documentation.

- ### Method
  - GET: Retrieve data.
  - POST: Create something new
  - PUT: Update data.

- DELETE: Delete data.

URL

- The URL for the endpoint you want to call.
  - Example:

http://apic/api/aaaLogin.xml

URL Parameters

- The parameters that you can pass as part of the URL.

Authentication

- Authentication type (Basic HTTP, token-based, and OAuth are common).
- Authentication credentials.

Additional HTTP Headers

- Additional HTTP headers required by the specific API.
- Example: Content-Type: application/json

Request Body

- JSON or XML containing the HTTP Message Body data bytes that are needed to complete the request.

Response Body

- JSON or XML containing the HTTP Message Body data bytes transmitted in an HTTP transaction message response.

this example below shows the REST API request and response process.

Figure 15-3 shows the REST API request and response process.

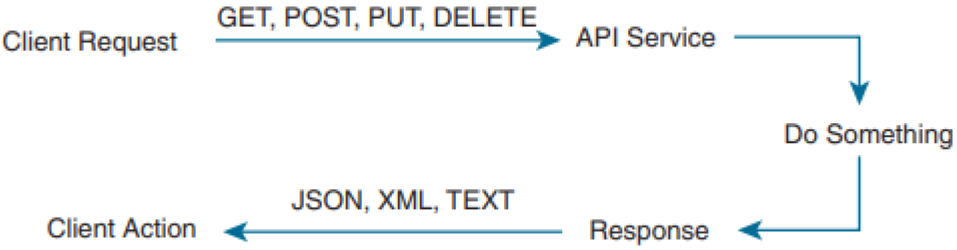


Figure 15-3 *REST API Request and Response*

Authentication

There are different types of authentication for REST APIs. Authentication is used to control access and access rights to the REST APIs. For example, some users might have read-only access, which means that they can use only the parts of the API that read data. Other users might have both read and write access. This means they can use the API to perform operations that not only read data but also add, edit, and delete data. These access rights are typically based on user-assigned roles such as Administrator that would have full rights to change data, whereas a plain User role might have read-only access rights.

The following list shows the types of authentication controls:

- **None:** The Web API resource is public; anybody can place a call. Generally, the case for GET methods, rarely for POST, PUT, DELETE.

- **Basic HTTP:** The username and password are passed to the server in an encoded string.
  - Authorization: Basic ENCODEDSTRING

- **Token:** A secret key generally retrieved from the Web API developer portal.
  - The keyword may change from one Web API to another: Bearer, token.
  - Passed with each API call

- **OAuth:** A sequence flow is initiated to retrieve an access token from an identity provider. The token is then passed with each API call.

- Open standard. User rights are associated with the token (OAuth scope).
- The token expires. It can be revoked. It can also be reissued via a refresh token.

Response

The returned data is defined in the Response portion, which includes the HTTP status codes
along with the data format and attributes.

HTTP status codes

- HTTP status codes are used to return success, error, or other statuses.(See http://www.w3.org/Protocols/HTTP/HTRESP.html.)

Some common examples are

- 200 OK
- 202 Accepted/Processing
- 401 Not Authorized

Content

- Often returned in different formats based on the request. Common formats are
JSON, XML, and Text.

NX-API

CLI commands are mostly run on the Nexus devices. NX-API enables you to access these CLIs
by making them available outside the switch by using HTTP/HTTPS. You can use this extension
to the existing Cisco Nexus CLI system on the Cisco Nexus 9000, 3000, 3500, 3600, and 7000
Series devices. NX-API supports show commands, configurations, and Linux Bash. NX-API uses
HTTP/HTTPS as its transport. CLIs are encoded into the HTTP/HTTPS POST body.

The NX-API back end 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
are capped. If the Nginx memory usage exceeds the cgroup limitations, the Nginx process is
restarted and restored.

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.
You should consider using HTTPS to secure your user’s login credentials.

NX-API provides a s**ession-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 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.

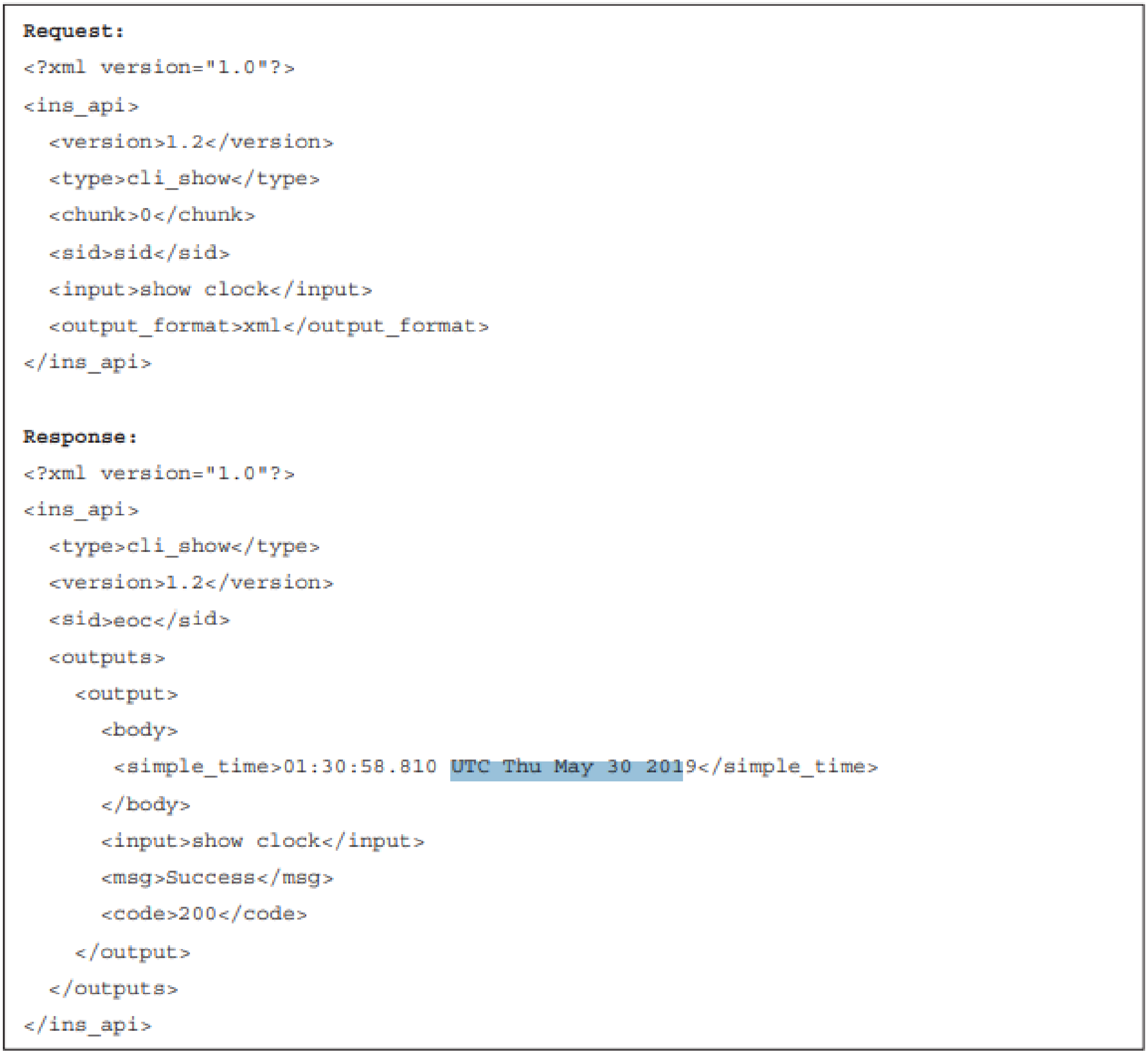
The commands, command type, and output type for the Cisco Nexus 9000 Series devices
are entered using NX-API by encoding the CLIs into the body of an HTTP/HTTPS POST.
The response to the request is returned in XML or JSON output format.

NX-API CLI is enabled by default for local access. **The remote HTTP access is disabled by default**
. First, **you need to enable the NX-API feature before you can send any API requests to the NX-OS software**
.

To enable the NX-API nxapi feature, enter these commands:

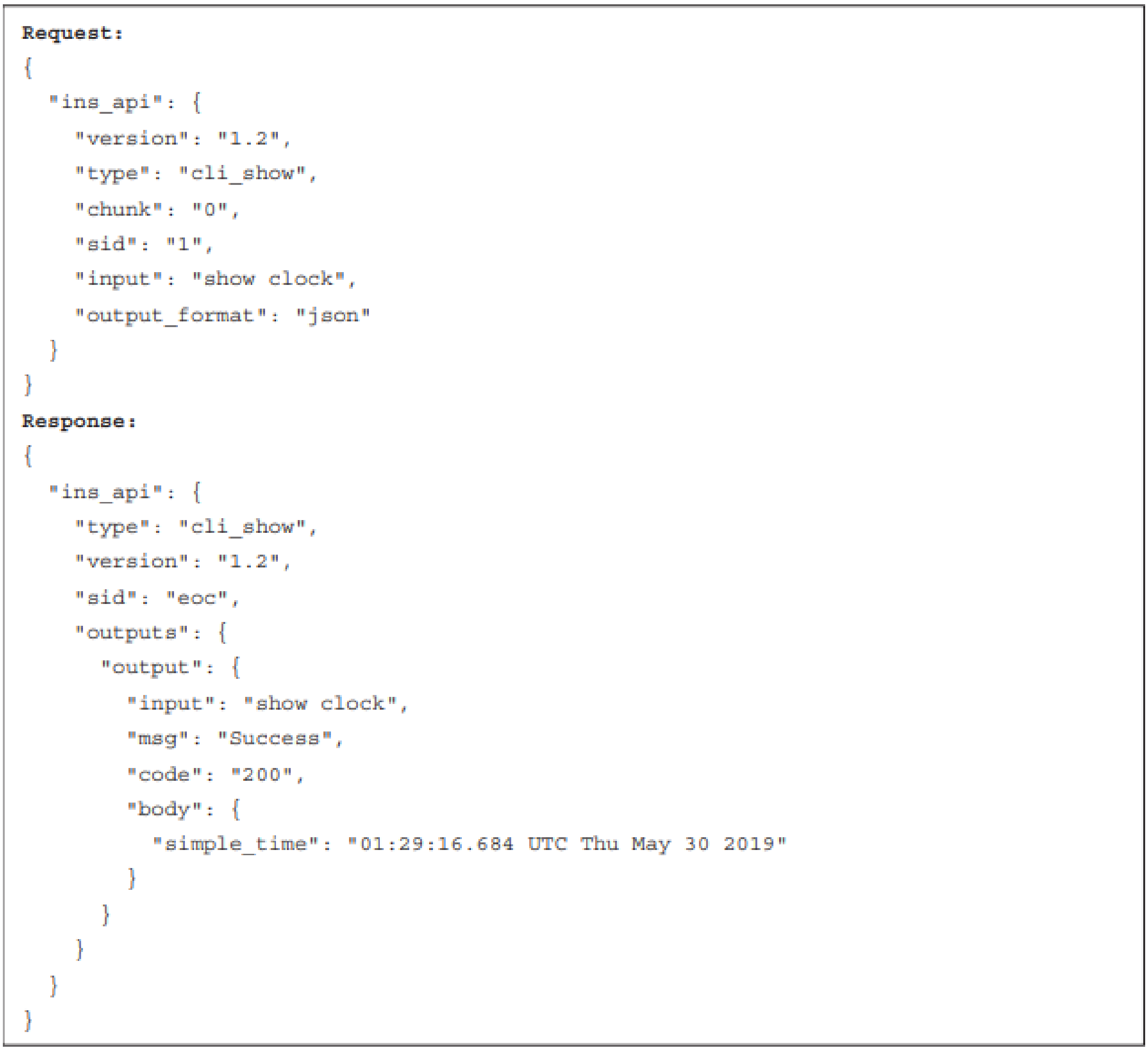
```
switch# conf t
switch(config)# feature nxapi
```

Example 15-18 *Request and Response in XML Format*



```
Request:
<?xml version="1.0"?>
<ins_api>
  <version>1.2</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>show clock</input>
  <output_format>xml</output_format>
</ins_api>

Response:
<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>1.2</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
       <simple_time>01:30:58.810 UTC Thu May 30 2019</simple_time>
      </body>
      <input>show clock</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

Example 15-19 *Request and Response in JSON Format*

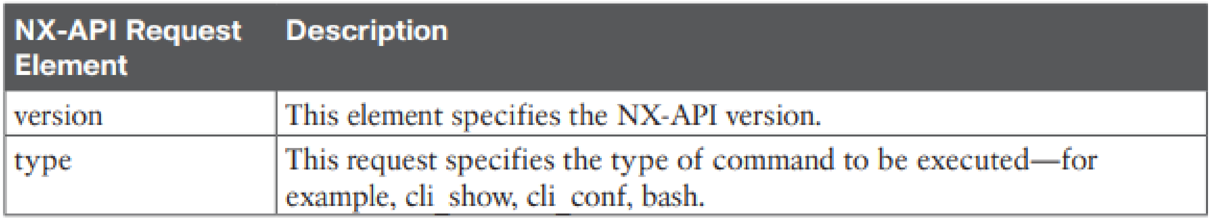


```
Request:
{
  "ins_api": {
    "version": "1.2",
    "type": "cli_show",
    "chunk": "0",
    "sid": "1",
    "input": "show clock",
    "output_format": "json"
  }
}
Response:
{
  "ins_api": {
    "type": "cli_show",
    "version": "1.2",
    "sid": "eoc",
    "outputs": {
      "output": {
        "input": "show clock",
        "msg": "Success",
        "code": "200",
        "body": {
          "simple_time": "01:29:16.684 UTC Thu May 30 2019"
        }
      }
    }
  }
}
```

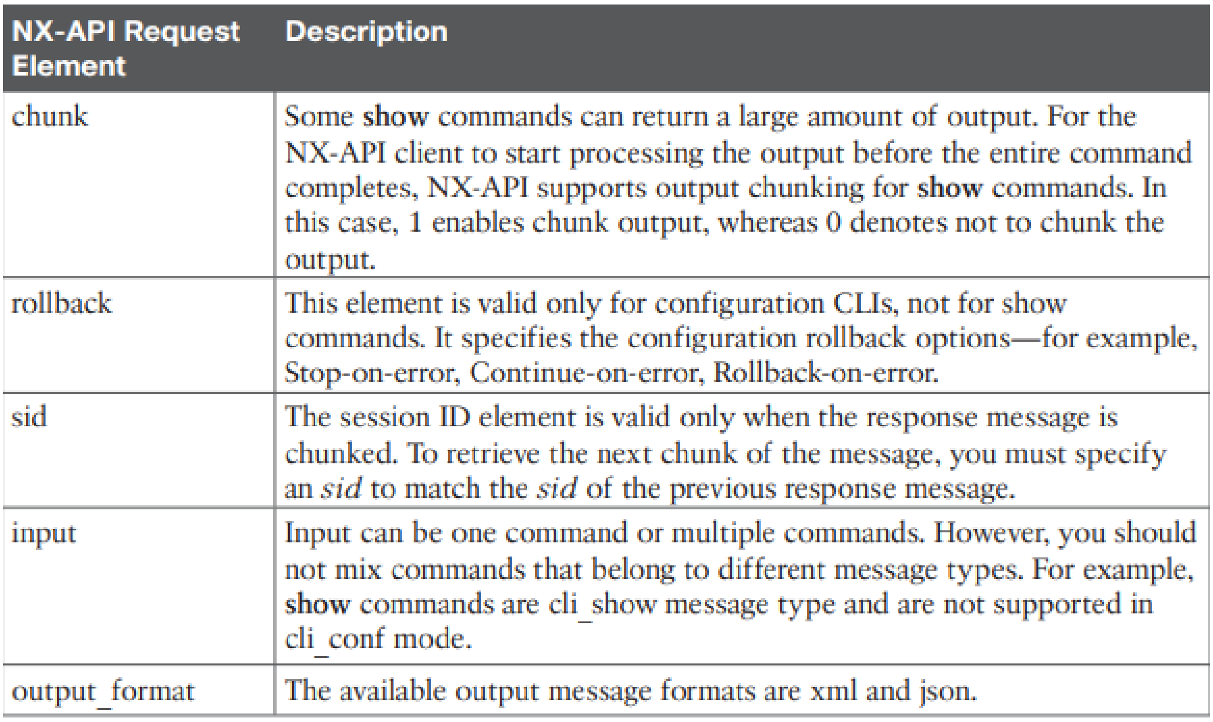
NX-API Request and Response Elements

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 can use the NX-API request elements shown in Table 15-9 to specify a CLI command
for XML or JSON format

Table 15-9 NX-API Request Elements



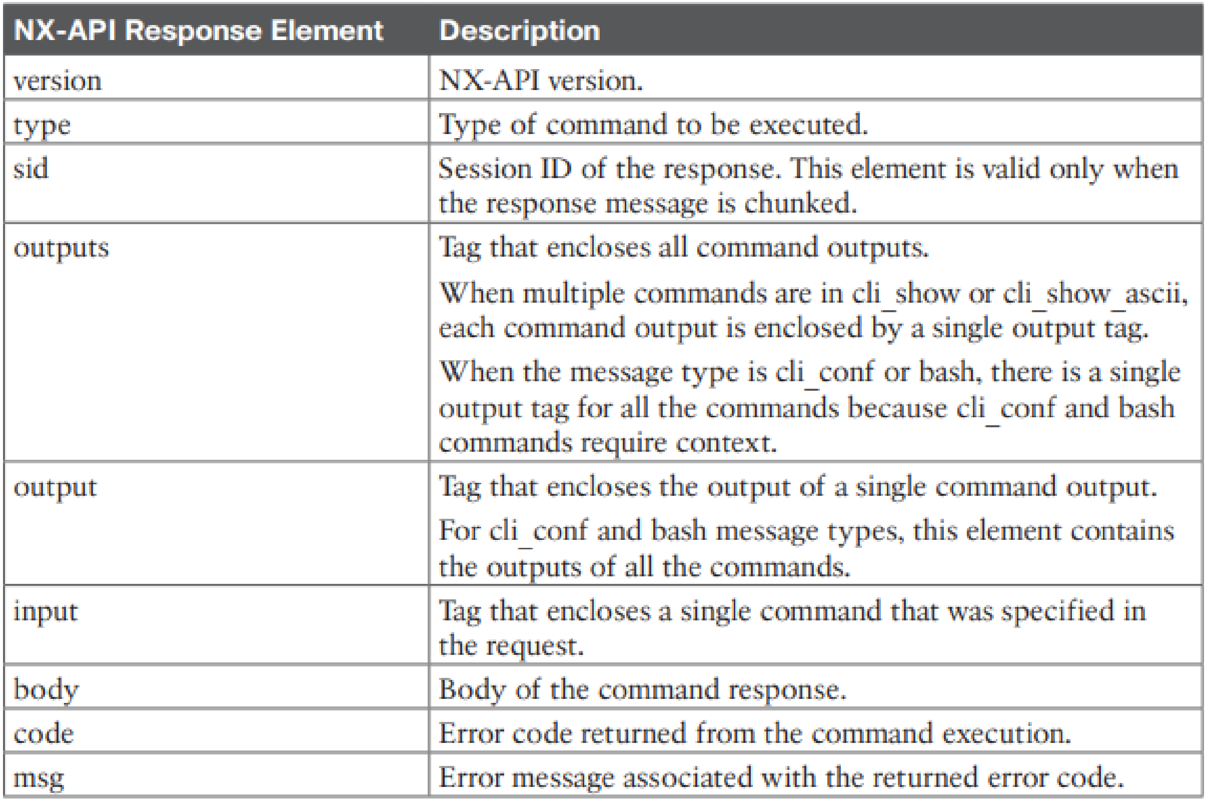
NX-API Request Element	Description
version	This element specifies the NX-API version.
type	This request specifies the type of command to be executed—for example, cli_show, cli_conf, bash.



NX-API Request Element	Description
chunk	Some **show** 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 **show** commands. In this case, 1 enables chunk output, whereas 0 denotes not to chunk the output.
rollback	This element is valid only for configuration CLIs, not for show commands. It specifies the configuration rollback options—for example, Stop-on-error, Continue-on-error, Rollback-on-error.
sid	The session ID element is valid only when the response message is chunked. To retrieve the next chunk of the message, you must specify an *sid* to match the *sid* of the previous response message.
input	Input can be one command or multiple commands. However, you should not mix commands that belong to different message types. For example, **show** commands are cli_show message type and are not supported in cli_conf mode.
output_format	The available output message formats are xml and json.

NX-API Response Element

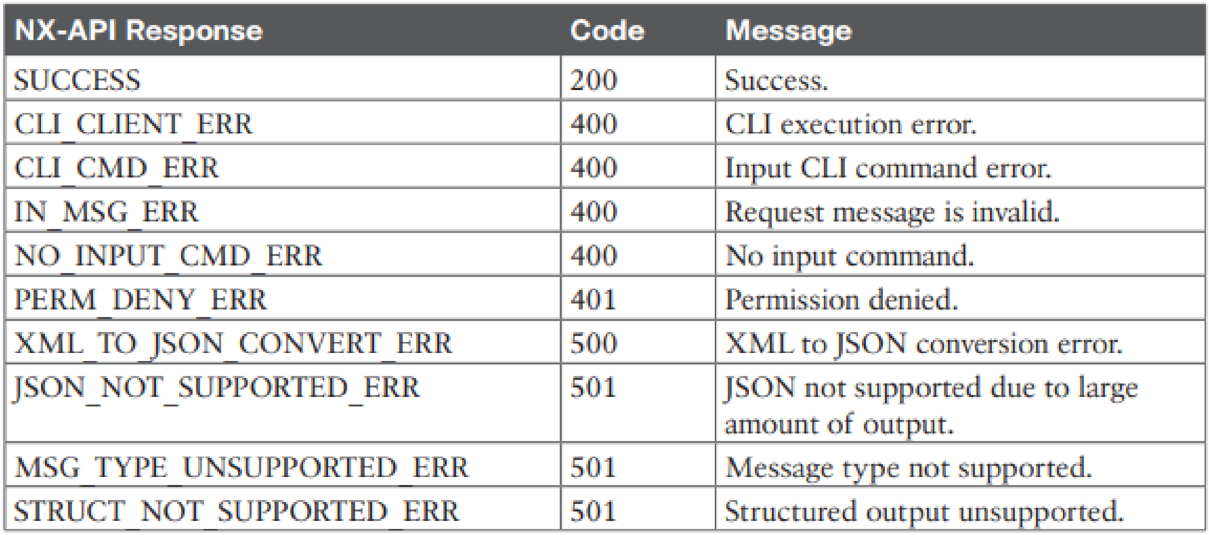
Table 15-10 NX-API Response Element



NX-API Response Element	Description
version	NX-API version.
type	Type of command to be executed.
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.
body	Body of the command response.
code	Error code returned from the command execution.
msg	Error message associated with the returned error code.

NX-API Response and Error Codes

Table 15-11 NX-API Response and Error Codes

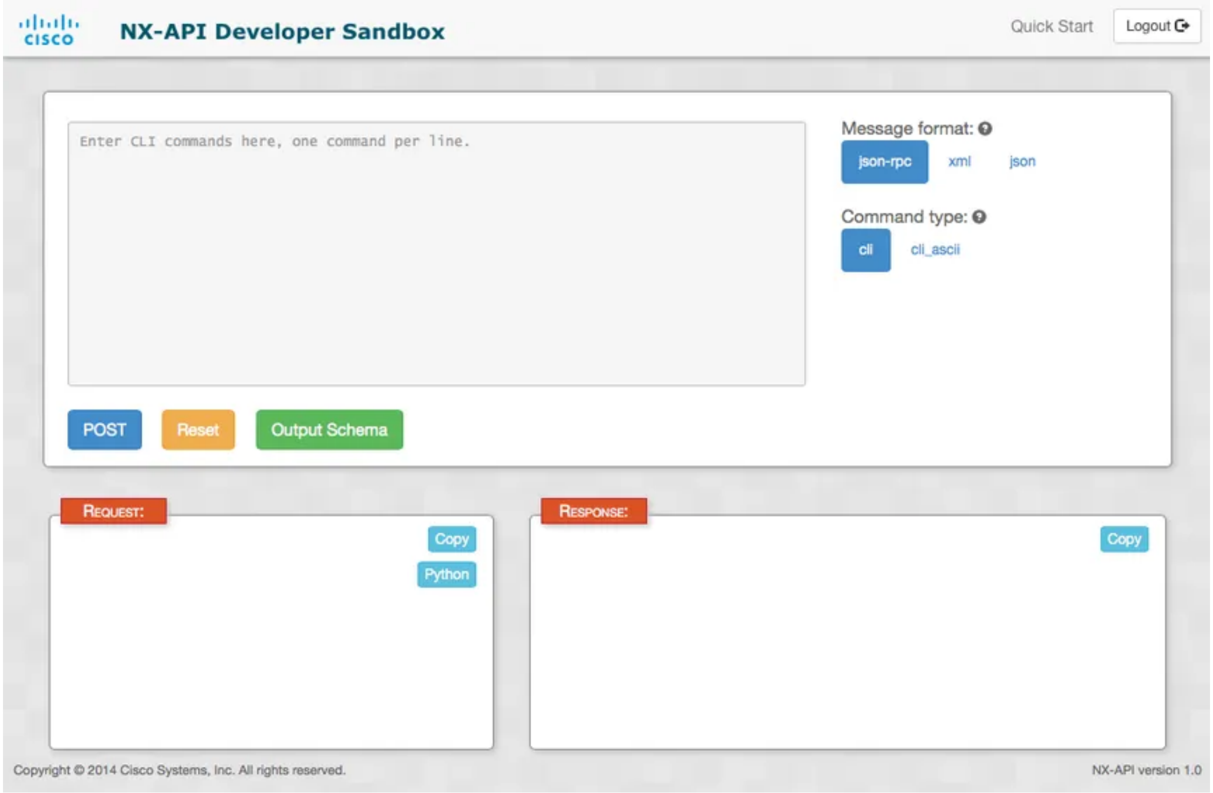


NX-API Response	Code	Message
SUCCESS	200	Success.
CLI_CLIENT_ERR	400	CLI execution error.
CLI_CMD_ERR	400	Input CLI command error.
IN_MSG_ERR	400	Request message is invalid.
NO_INPUT_CMD_ERR	400	No input command.
PERM_DENY_ERR	401	Permission denied.
XML_TO_JSON_CONVERT_ERR	500	XML to JSON conversion error.
JSON_NOT_SUPPORTED_ERR	501	JSON not supported due to large amount of output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
STRUCT_NOT_SUPPORTED_ERR	501	Structured output unsupported.

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 **(bottom-left pane)**, and Response **(bottom-right pane)**



The Request pane also has a series of tabs. Each tab represents a different language: Python, Java, and JavaScript. 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.

Controls in the Command pane enable you to choose a supported command type, such as cli_show, cli_show_ascii, cli_conf, and a message format, such as XML or JSON. The available options vary depending on the chosen method.

When you type or paste one or more CLI commands into the Command pane, **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), the Response pane
displays the API response.

Evaluate Automation and Orchestration Technologies

Most of the early automation tools were developed for server automation. As use cases of
automation increased, many companies developed products that supported automation from
day one. Cisco is no different; it developed several products such as the Cisco UCS Server,
Cisco Nexus switches, and Cisco ACI, which now support automation tools such as Ansible
and Puppet. Automation tools help maintain consistent configuration throughout a network
with no or minimal human intervention. Some automation tools are agentless,
such as Ansible, which utilizes device-specific APIs or SSH to push configuration to network devices,
and do not require installation of an agent. However, some automation tools, such as Puppet,
work with the help of an agent, meaning they are installed on the network devices, which are
responsible for converting configuration details to a device-specific configuration.

Ansible

Ansible is an agentless configuration management or orchestration tool. Users have the
flexibility to turn their laptops into an Ansible control station to automate basic tasks, or
they can deploy a dedicated host to use Ansible as an orchestration tool to roll

out application updates while ensuring minimal downtime. Ansible provides a simple domain-specific
language (DSL) to enable these different use cases. Ansible is popular among infrastructure
engineers and developers because it requires minimal time and effort to get up and running.

how Ansible works :

A basic workflow for Ansible using playbooks looks something like that shown in below

1. Engineers create Ansible playbooks in YAML that describe a workflow or the configuration of infrastructure.
2. Ansible playbooks are deployed to an Ansible control station.
3. When the control station runs the Ansible playbooks, they typically copy modules
   written in Python to remote hosts.
4. Finally, Ansible runs the modules on the remote hosts to perform the work described
   in playbooks.

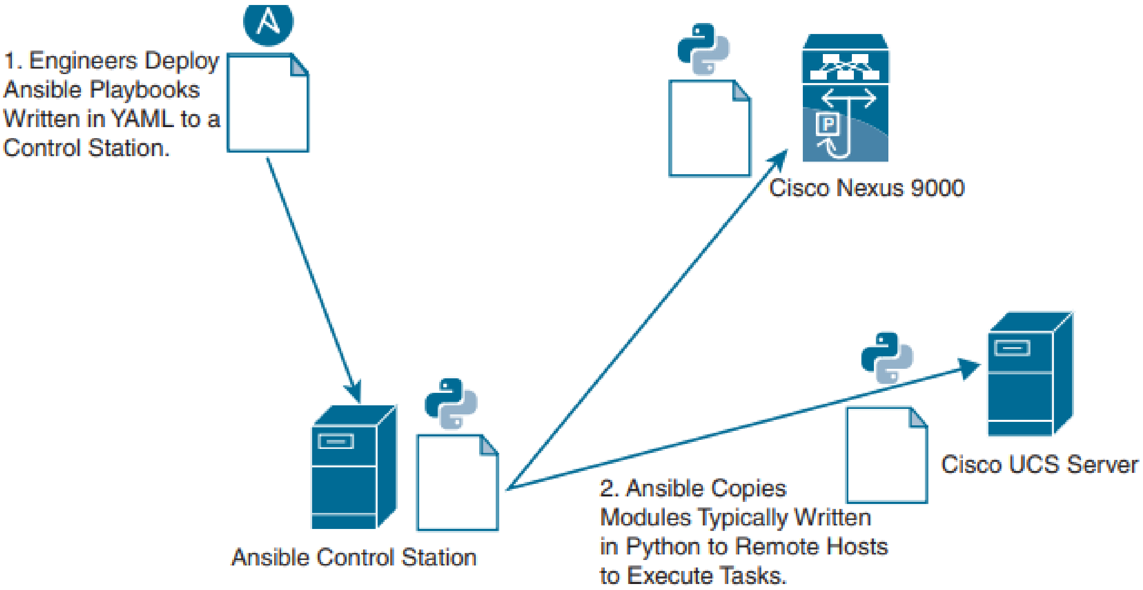

1. Engineers Deploy Ansible Playbooks Written in YAML to a Control Station.
Cisco Nexus 9000
Cisco UCS Server
Ansible Control Station
2. Ansible Copies Modules Typically Written in Python to Remote Hosts to Execute Tasks.

Figure 16-1 *Workflow for Ansible Using Playbooks*

Ansible Components

Ansible requires a control machine to run the Ansible tool. By default, Ansible uses a push model to push changes to remote hosts from the Ansible control machine. The control machine can be any Linux/UNIX host with a Python interpreter that supports SSH or the required transport to devices managed by Ansible. Some of the important components of the Ansible control machine are as follows:

- **Modules** are typically written in Python. They are typically copied to remote hosts and run by the Ansible tool. Ansible modules are referenced as tasks in Ansible playbooks or using CLI arguments in the Ansible ad hoc CLI tool.

- **Inventory files** contain the hosts operated by Ansible. They contain group and host definitions that can be referenced by Ansible playbooks or using CLI arguments from the Ansible ad hoc CLI tool. A host can belong to multiple groups.

- **Playbooks** are written in YAML and contain Ansible domain-specific language. To enable reuse, playbooks can be modularized much like software. Variables containing data for playbooks can be separated into YAML files residing on the Ansible control machine.

- **Configuration** files control how the tool runs. For example, the configuration file can change the default directories of the modules.

NOTE: the ansible ADhock tool is an ansible command line toole that can be used to execute commands the ansible python scripts

Ansible Cli Tool : this is commonly referred as AD_hoc tol

ansible-playbook : this runs ansible playbooks against the targeted ansible hosts ,and the playbooks **contain the dsl (Domain Specific Language) in YAML format .**

ansible-vault: if the playbook requires a sensitive data that the engineers doesn't want to expose in a plain text format , the ansible-vault comes to play which means it is a measure of security .

ansible-pull: enable clients to pull modules from a centralized server .

ansible-docs: enables engineers to pars docs strings of ansible modules to see examples syntax and the parameters module require.

ansible-galaxy: can be used to create and download roles from the ansible community (the ansible galaxy is public repository of ansible playbooks grouped into roles from the community)

steps for setting the environment for ansible :

Step one : we first create a directory on our local machine and then we clone the github repository inside the created directory .

NOTE: on NX_OS we can only enable the **feature NX_API and ssh** using the command :

```
feature nxapi
```

Step Two: now we move to creating the **host file or Inventory-file**

we can create variables for the username and password

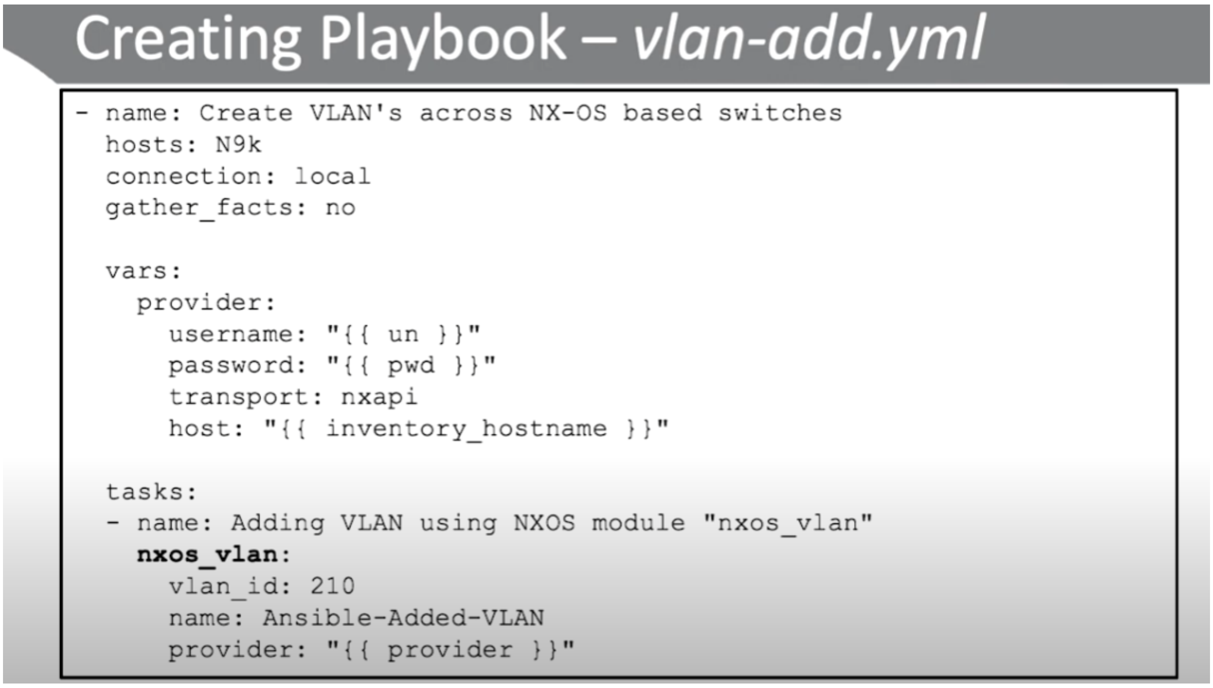
we can specify the hostnames or the ip addresses under one of the work groups in our example the work group is **named N9K**

Step Three: testing the ansible execution , to test the ansible execution against the inventory file example to perform a ping test :

the command is like this from your centralized station:

```
ansible -i host-file (or inventory file ) N9K(the work group
```

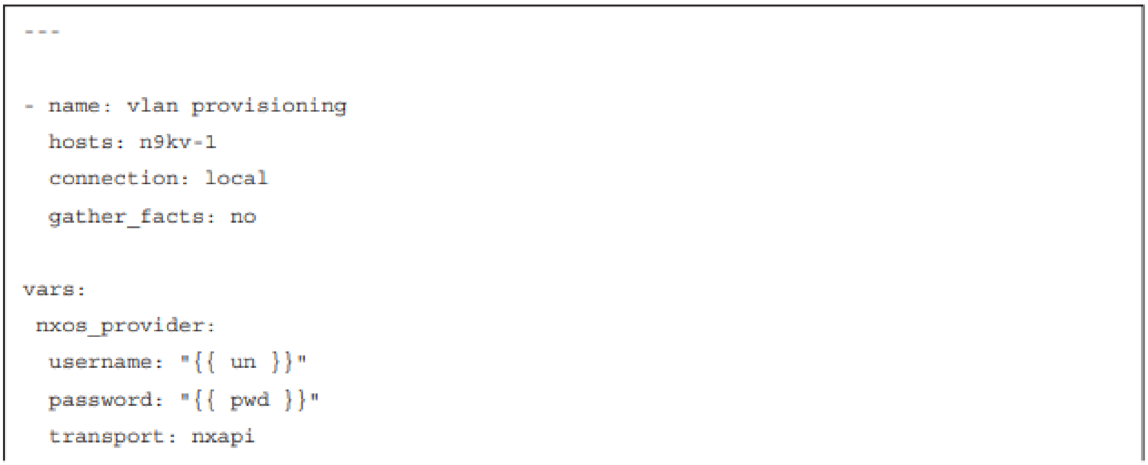
Step Four : after testing the connectivity or the execution of ansible commands we can then start creating ansible playbooks



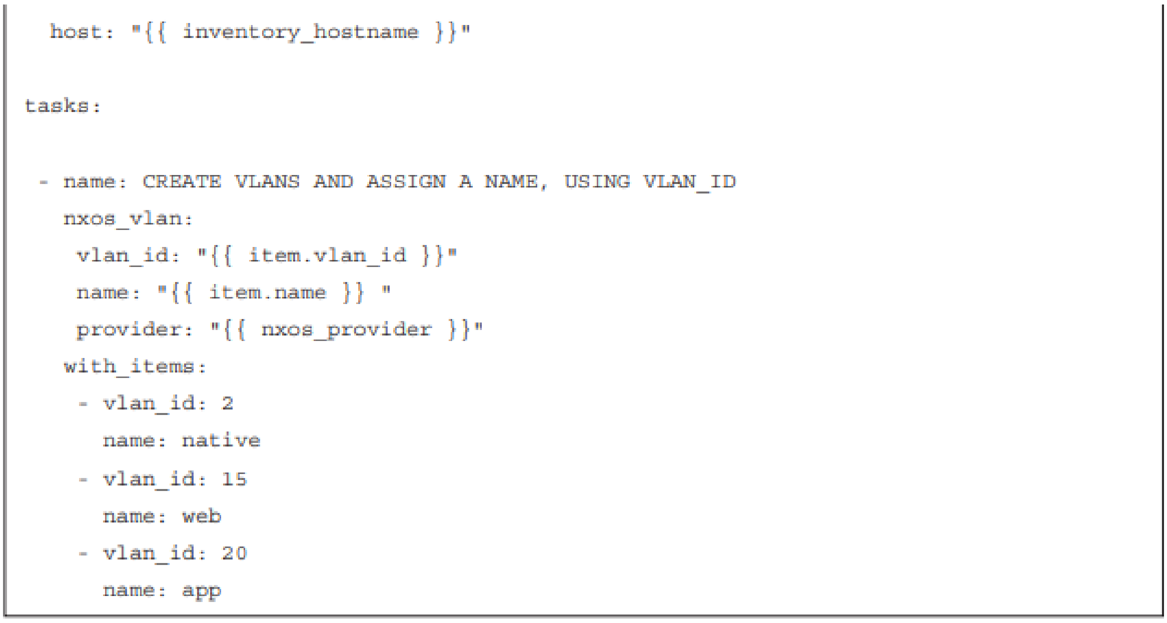
Creating Playbook – *vlan-add.yml*

```
- name: Create VLAN's across NX-OS based switches
  hosts: N9k
  connection: local
  gather_facts: no

  vars:
    provider:
      username: "{{ un }}"
      password: "{{ pwd }}"
      transport: nxapi
      host: "{{ inventory_hostname }}"

  tasks:
  - name: Adding VLAN using NXOS module "nxos_vlan"
    nxos_vlan:
      vlan_id: 210
      name: Ansible-Added-VLAN
      provider: "{{ provider }}"
```

Example 16-1 *Ansible Playbook Example*



```
---

- name: vlan provisioning
  hosts: n9kv-1
  connection: local
  gather_facts: no

vars:
 nxos_provider:
  username: "{{ un }}"
  password: "{{ pwd }}"
  transport: nxapi
```



```
  host: "{{ inventory_hostname }}"

tasks:

 - name: CREATE VLANS AND ASSIGN A NAME, USING VLAN_ID
   nxos_vlan:
    vlan_id: "{{ item.vlan_id }}"
    name: "{{ item.name }} "
    provider: "{{ nxos_provider }}"
   with_items:
    - vlan_id: 2
      name: native
    - vlan_id: 15
      name: web
    - vlan_id: 20
      name: app
```

The playbook has these fields:

■ **The name:** field defines the playbook name.

■ **The hosts:** n9kv-1 field specifies the set of hosts that will be configured by the playbook.

■ **The connection:** local field denotes that the task will be handled by Ansible, just like a local action.

■ **The gather_facts:** no field denotes that no information from the device will be collected.

■ **The tasks:** field specifies the task that will be run on the Nexus device.

■ **The vars**: field defines the username and password and transport method to achieve the tasks at hand. In this example, the configuration will be done using NXAPI as the transport method.

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.**

this example below illustrates how Puppet automation can be integrated with Cisco Nexus and UCS devices.

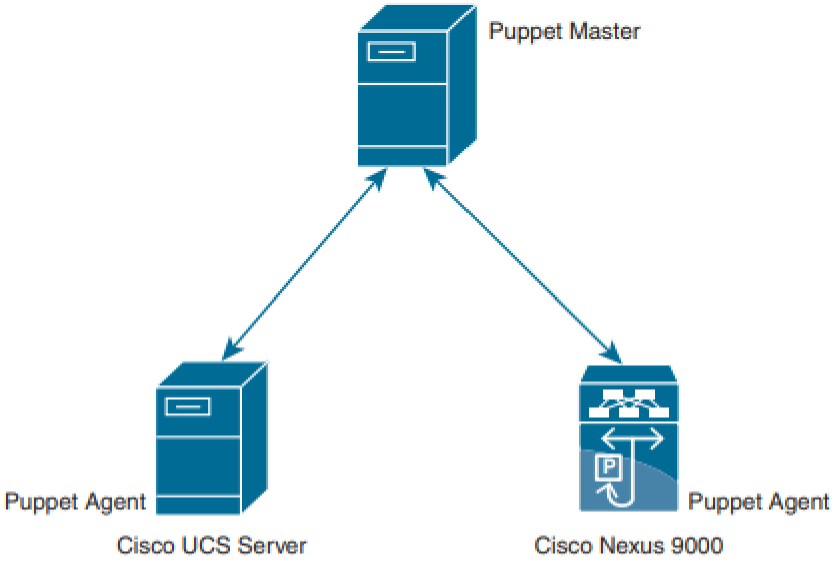


Figure 16-2 *Puppet Overview*

Puppet Workflow

1. **Define:** With Puppet's declarative language, you design a graph of a relationship
   between resources within reusable modules called manifests. The manifests define
   your infrastructure in its desired state.
2. **Simulate:** Using manifests, Puppet simulates deployments, enabling you to test
   changes without disruption to your infrastructure.
3. **Enforce:** Puppet compares your system to the desired state as you define it, and automatically enforces it to the desired state, ensuring your system is in compliance.
4. **Report:** The Puppet Dashboard reports back the relationship between components
   and all changes. And with the open API, you can integrate Puppet with third-party
   monitoring tools.

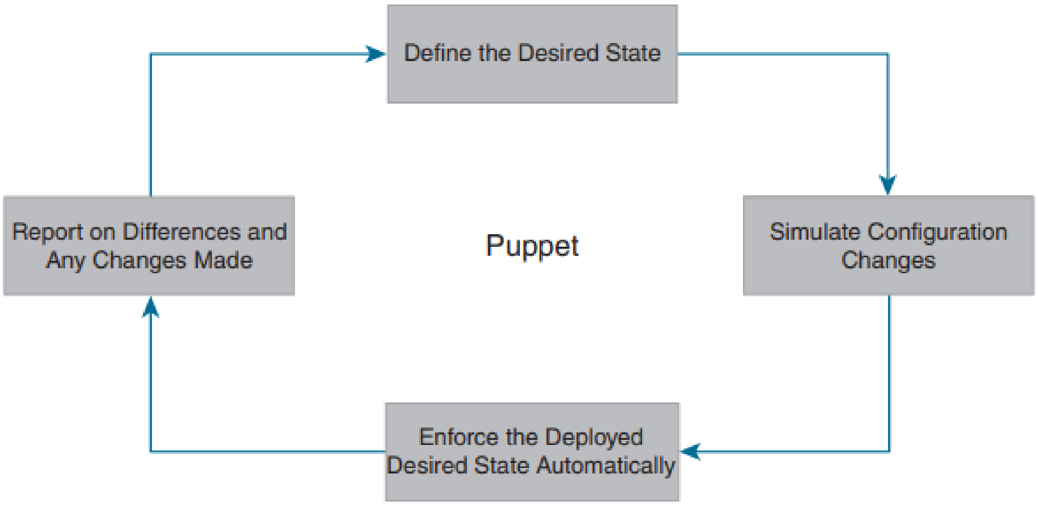

Define the Desired State
Simulate Configuration Changes
Puppet
Enforce the Deployed Desired State Automatically
Report on Differences and Any Changes Made

Figure 16-3 *Puppet Workflow*

- Puppet is an open source cross platform system
- it is a declarative language for describing system configuration
- Client/Server based Application

  - Server = Puppetmaster

  - Client = node or puppet
- Puppet is idempotent

  - Detects current state of the system

  - Enforces only new configuration to the system

Components:

- Manifests are files containing Puppet's declarative language

  - Helps define relationships between resources within reusable modules
- Core of Puppet language is declaring resources

  - if there is a dependency of one resource on another, the relationship should be explicitly stated .
- Class is a set of common configurations - resources , variables , ...etc .
- Modules - Collecation of files and directions containing Puppet manifests .
- Hierarchy

- Modules { Manifeast{ Classes{ Resources } ... }... }

Puppet and NX-OS Environment Integration

The ciscopuppet module allows a network administrator to manage Cisco Nexus network elements using Puppet. The Puppet agent is supported on Cisco Nexus 3000, 5000, 6000, 7000, and 9000 Series of switches. The Puppet agent can be installed on various NX-OS environments as shown in Table

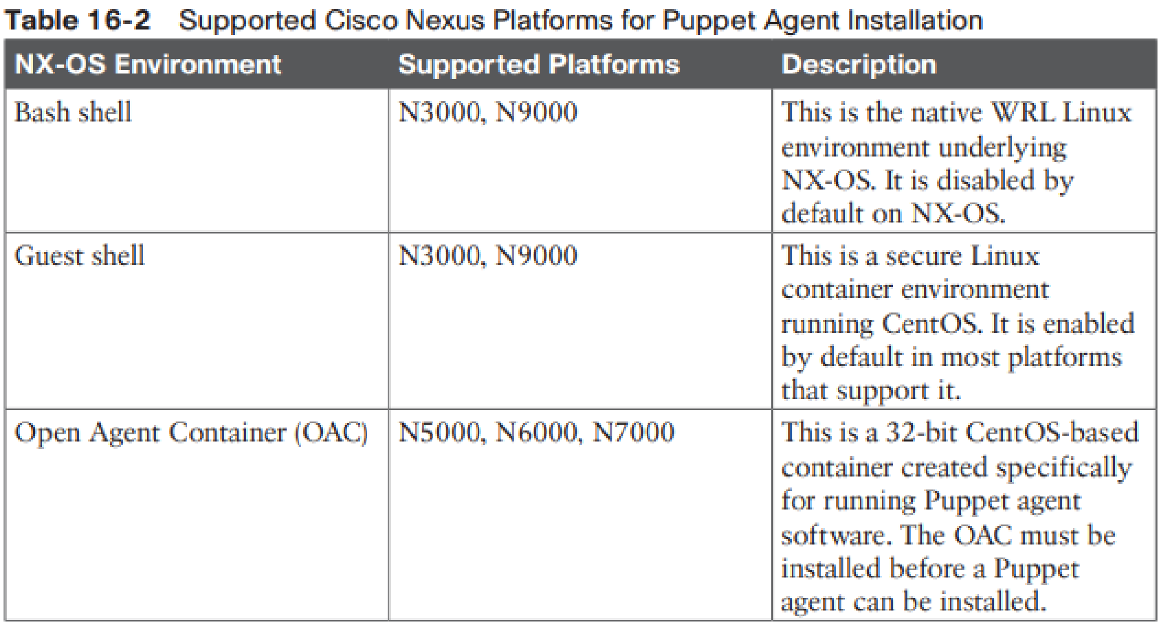


Table 16-2 Supported Cisco Nexus Platforms for Puppet Agent Installation

NX-OS Environment	Supported Platforms	Description
Bash shell	N3000, N9000	This is the native WRL Linux environment underlying NX-OS. It is disabled by default on NX-OS.
Guest shell	N3000, N9000	This is a secure Linux container environment running CentOS. It is enabled by default in most platforms that support it.
Open Agent Container (OAC)	N5000, N6000, N7000	This is a 32-bit CentOS-based container created specifically for running Puppet agent software. The OAC must be installed before a Puppet agent can be installed.

NOTE: Starting in NX-OS release 9.2(1) and onward, installing a Puppet agent in the Bashshell hosting environment is no longer supported. Instead, the Puppet agent software should be installed on the Guest shell hosting environment.

this is the manual steps for installing puppet on a cisco nx-os device

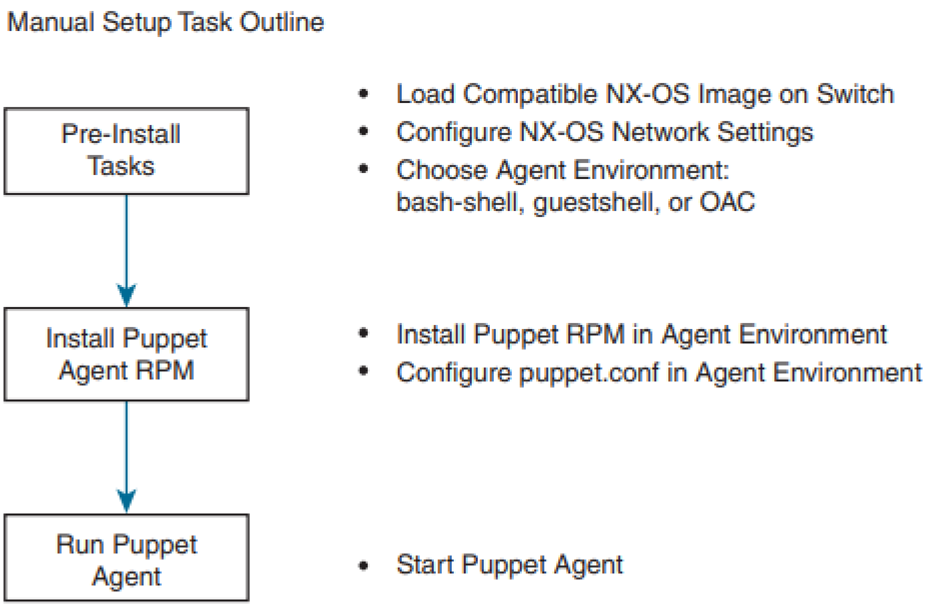

Manual Setup Task Outline
Pre-Install Tasks
Install Puppet Agent RPM
Run Puppet Agent
Load Compatible NX-OS Image on Switch
Configure NX-OS Network Settings
Choose Agent Environment: bash-shell, guestshell, or OAC
Install Puppet RPM in Agent Environment
Configure puppet.conf in Agent Environment
Start Puppet Agent

Figure 16-4 *Manual Puppet Installation Steps on Cisco Nexus Switches*

For managing Cisco devices using Puppet agents, the Puppet master needs to install the

ciscopuppet module

. You can use the following command to install the ciscopuppet module on the Puppet master:

```
puppet module install puppetlabs-ciscopuppet
```

Puppet Agent Installation

This section is common to Bash shell, Guest shell, and the Open Agent Container. The

following steps guide you through the installation process of the Puppet agent on Cisco

NX-OS software:

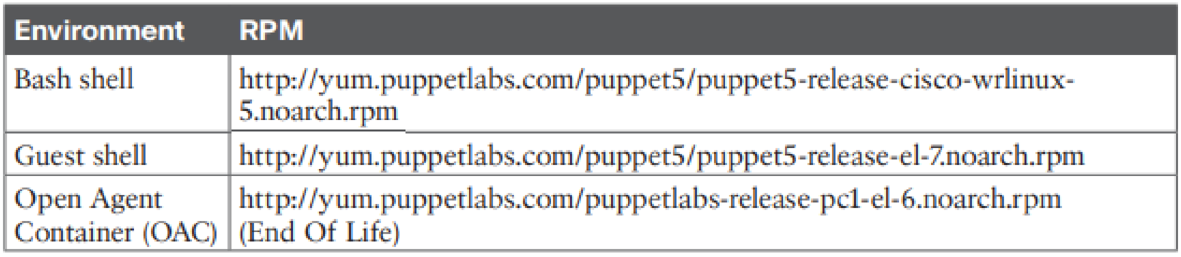
Step 1.

- Select and install the Puppet agent RPM. Here, you import the Puppet GPG keys and install the appropriate Puppet RPM for your agent environment. Install GPG Keys:
  rpm --import
  http://yum.puppetlabs.com/RPM-GPG-KEY-puppetlabs
  rpm --import

http://yum.puppetlabs.com/RPM-GPG-KEY-reductive
rpm --import
http://yum.puppetlabs.com/RPM-GPG-KEY-puppet

the table below shows RPM URLs for the different environments on NX-OS.

Table 16-3 RPM URLs For Different Environments on NX-OS



Environment	RPM
Bash shell	http://yum.puppetlabs.com/puppet5/puppet5-release-cisco-wrlinux-5.noarch.rpm
Guest shell	http://yum.puppetlabs.com/puppet5/puppet5-release-el-7.noarch.rpm
Open Agent Container (OAC)	http://yum.puppetlabs.com/puppetlabs-release-pc1-el-6.noarch.rpm (End Of Life)

- Install RPM using the following command:

```
yum install $PUPPET_RPM
yum install puppet
```

- where $PUPPET_RPM is the URL from Table .
  Update the path variable using the following command:
  export
  PATH=/opt/puppetlabs/puppet/bin:/opt/puppetlabs/puppet/lib:$PATH

Step 2.

- Add the Puppet server name to the configuration file at /etc/puppetlabs/puppet/puppet.conf:
  [main]
  server =
  mypuppetmaster.mycompany.com
  certname = this_node.mycompany.com
  where this_node is the host name of the Nexus device and
  mycompany.com is
  the domain name configured under VRF management.

Step 3.

- Install the cisco_node_utils gem.
  The cisco_node_utils Rubygem is a required component of the ciscopuppet module. This gem contains platform APIs for interfacing between Cisco CLI and Puppet agent resources. The Puppet agent can automatically install the

gem by simply using the ciscopuppet::install helper class, or it can be installed
manually. The following command installs cisco_node_utils manually:

```
gem install cisco_node_utils
```

Step 4.

- Run the Puppet agent.
  Executing the puppet agent command (with no arguments) will start the Puppet agent process with the default run interval of 30 minutes. Use the -t option
  to run the Puppet agent in test mode, which runs the agent a single time and
  stops:

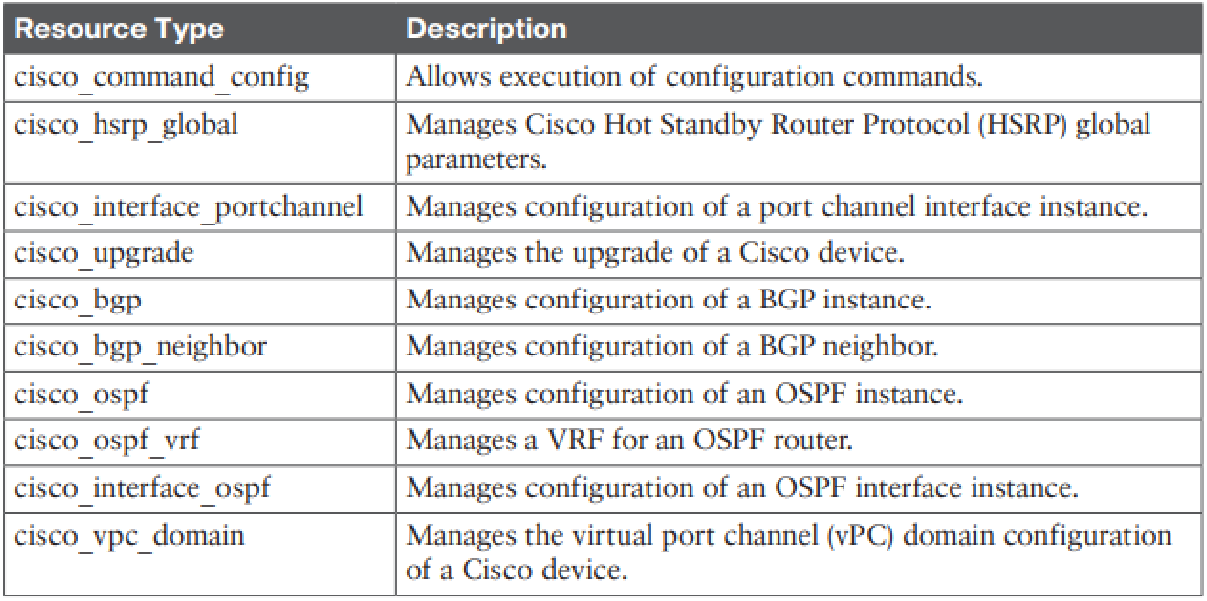
```

puppet agent -t
```

The Cisco Nexus network elements and operating systems managed by this Puppet module
are continuously expanding. This GitHub repository contains the latest version of the
ciscopuppet module source code. Supported versions of the ciscopuppet module are
available at Puppet Forge.

Resource Types

Puppet has predefined resource types that can be used to configure features on NX-OS.
Some of the resource types are shown in Table

Table 16-4 Puppet Resource Types



Resource Type	Description
cisco_command_config	Allows execution of configuration commands.
cisco_hsrp_global	Manages Cisco Hot Standby Router Protocol (HSRP) global parameters.
cisco_interface_portchannel	Manages configuration of a port channel interface instance.
cisco_upgrade	Manages the upgrade of a Cisco device.
cisco_bgp	Manages configuration of a BGP instance.
cisco_bgp_neighbor	Manages configuration of a BGP neighbor.
cisco_ospf	Manages configuration of an OSPF instance.
cisco_ospf_vrf	Manages a VRF for an OSPF router.
cisco_interface_ospf	Manages configuration of an OSPF interface instance.
cisco_vpc_domain	Manages the virtual port channel (vPC) domain configuration of a Cisco device.

Sample Manifest: OSPF

The following example demonstrates how to define a manifest that uses ciscopuppet to
configure OSPF on a Cisco Nexus switch. Three resource types are used to define an OSPF
instance, basic OSPF router settings, and OSPF interface settings:

- **cisco_ospf**
- **cisco_ospf_vrf**
- **cisco_interface_ospf**

The first manifest type defines the router instance using cisco_ospf. The title "Sample"
becomes the router instance name:

```
cisco _ ospf {"Sample":
ensure => present,
}
```

The next type to define is cisco_ospf_vrf. The title includes the OSPF router instance name and the VRF name. Note that a non-VRF configuration uses “default” as the VRF name:

```
cisco _ ospf _ vrf {"Sample default":
ensure => 'present',
default _ metric => '5',
auto _ cost => '46000',
}
```

Finally, you define the OSPF interface settings. The title here includes the interface name and the OSPF router instance name:

```

cisco _ interface _ ospf {"Ethernet1/5 Sample":
ensure => present,
area => 100,
cost => "100",
}
```

Puppet and Cisco UCS Manager Integration

The Cisco Puppet module for UCSM allows administrators to automate all aspects of Cisco UCS management, including server, network, storage, and hypervisor management. The bulk of the Cisco UCSM Puppet module works on the UCS Manager’s Management Information Tree (MIT), performing create, modify, or delete actions on the managed objects (MOs) in the tree. The ucsm module has a dependency on the ucsmsdk Python library

Example 16-2 *Puppet Manifest Example*

```
ucsm_vlan{'fabricVlan':
policy_name => "vlan603",
id => "603",
default_net => "yes",
ip => "192.168.10.132",
username => "admin",
password => "password",
state => "present",
}
```

- **ucsm_vlan:** The VLAN resource type defined in the Puppet DSL. This is required to identify which resource you intend to configure.
- **policy_name:** This is the name of the policy to be configured.
- **default_net:** If the newly created VLAN is a native VLAN, this parameter has to be set to “yes”. Otherwise, it should be set to “no”.
- **id:** This is the range of VLAN IDs (for example, “2009-2019”, “29,35,40-45”, “23”, “23,34-45”).
- **ip:** This is the IP address of the UCS server.
- **username:** This is the administrative username.
- **password:** This is the administrative password.
- **state:** This parameter ensures whether the policy should be present or absent on the UCS server.

Python

Python is a programming language that has high-level data structures and a simple approach to object-oriented programming. Python’s 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 Cisco Nexus Series devices support Python version 2.7.5 in both interactive and noninteractive (script) modes. Python is also supported in the Guest Shell.

The Python scripting capability provides programmatic access to the device’s 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 scripting capability on Cisco Nexus switches enables you to perform the following tasks:

- Run a script to verify configuration on switch bootup
- Back up a configuration
- Perform proactive congestion management by monitoring and responding to buffer utilization characteristics
- Perform integration with the PowerOn Auto Provisioning or EEM modules
- Perform a job at a specific time interval (such as Port Auto Description)
- Programmatically access the switch command-line interface to perform various tasks

Python Package for Cisco

Cisco NX-OS provides a Python package named cisco package that enables access to many core network device modules, such as interfaces, VLANs, VRFs, ACLs, and routes. After you have imported the cisco package, you can display its help by entering help (cisco) at the Python prompt. To display help on a specific module in the Cisco Python package, enter help (cisco.module_name), where module_name is the name of the module. For example, to display help on the Cisco ACL module, enter help (cisco.acl).

the example below shows how to display information about the Python package for Cisco .

The methods and functions in the Python package named cisco are implemented in Python source files included with the software development kit (SDK) package. Many of these files have documentation embedded within the source code. In the Python source files, documentation is contained in documentation strings, bracketed by three backticks, or ```. Some source files and methods are for internal use and do not have embedded documentation.

the table below shows the functionality that the cisco package provides.

From the cisco package, you can import individual modules as needed using from cisco.module _ name import * where module_name is the name of the individual module. In the example, the ACL module is imported.

```
import cisco
from cisco.acl import *
```

Other useful modules include the cli package and the json package. The cli package is used to allow Python scripts running on the NX-OS device to interact with the CLI to get and set configuration on the device. This library has one function within it named cli. The input parameters to the function are the CLI commands the user wants to run, and the output is a string representing the parser output from the CLI command.

```
from cli import *
import json
```

After starting Python and importing the required packages and modules, you can run Python scripts directly, or you can enter blocks of Python code and run the code.

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.

You must 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:

1. cli() returns the raw output of CLI commands, including control or special characters. The interactive Python interpreter prints control or special characters “escaped.”
   A carriage return is printed as ‘\n’ and gives results that can be difficult to read.
   The clip() API gives results that are more readable.
   Example:

```
string = cli ("cli-command")
```

1. clid() returns JSON output for the CLI command, if XML support exists for the command; otherwise, an exception is thrown. This API can be useful when searching the
   output of show commands.
   Example:

```
json_string = clid ("cli-command")
```

1. clip() prints the output of the CLI command directly to stdout and returns nothing to
   Python.
   Example:

```
clip ("cli-command")
```

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:

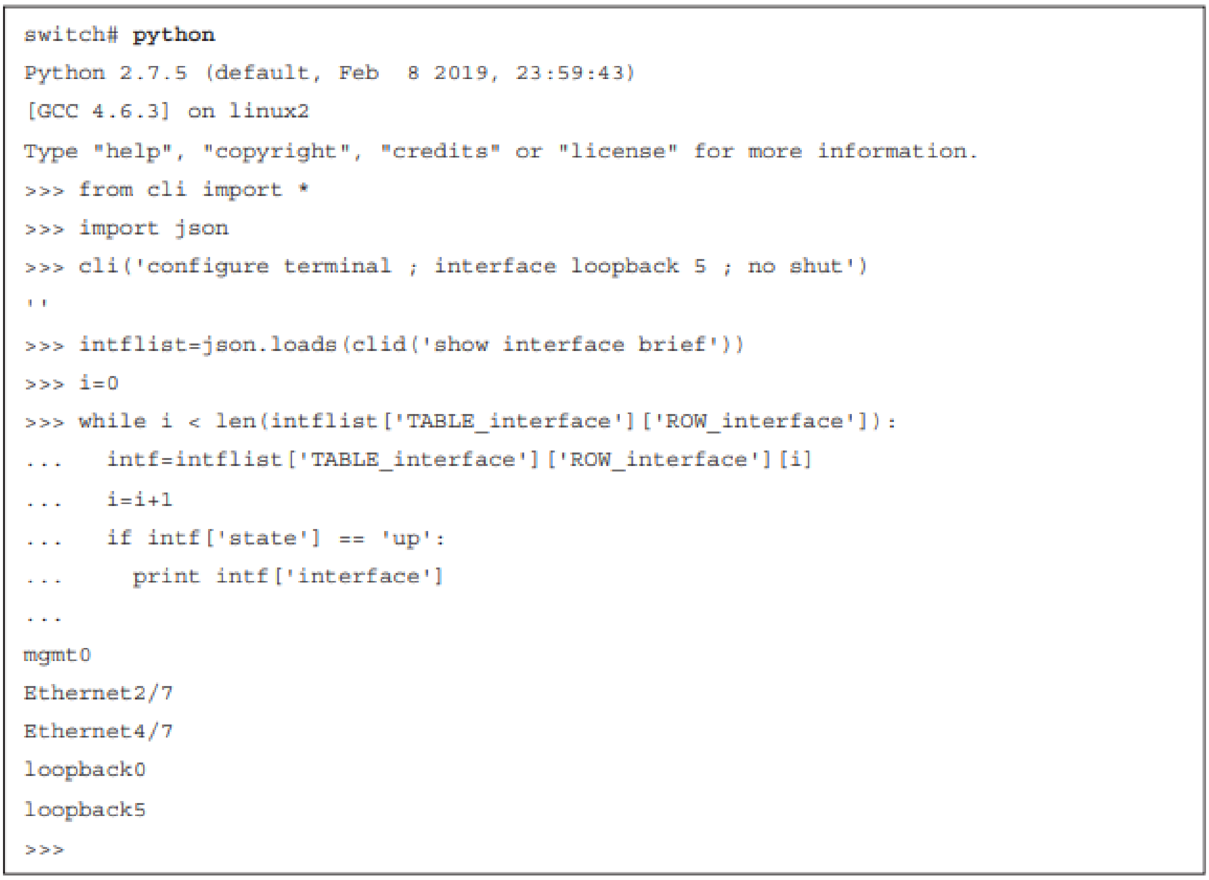
```
cli("conf t ; interface eth4/1 ; shut")
```

Python in Interactive Mode

To enter the Python shell, enter the python command from the NX-OS command line with
no parameters. You can enter lines of Python code to execute a block of code. A colon (:)
at the end of a line tells the Python interpreter that the subsequent lines will form a code
block. After the colon, you must indent the subsequent lines in the block, following Python
indentation rules. After you have typed the block, press Return or Enter twice to execute
the code.

this example below shows how to invoke Python from the CLI and run Python commands
interactively.

Example 16-4 *Interactive Python Example*



```
switch# python
Python 2.7.5 (default, Feb  8 2019, 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(clid('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
>>>
```

The preceding example brings the loopback 5 interface UP and shows how to query the
interfaces running on the switch.

Python in Noninteractive Mode

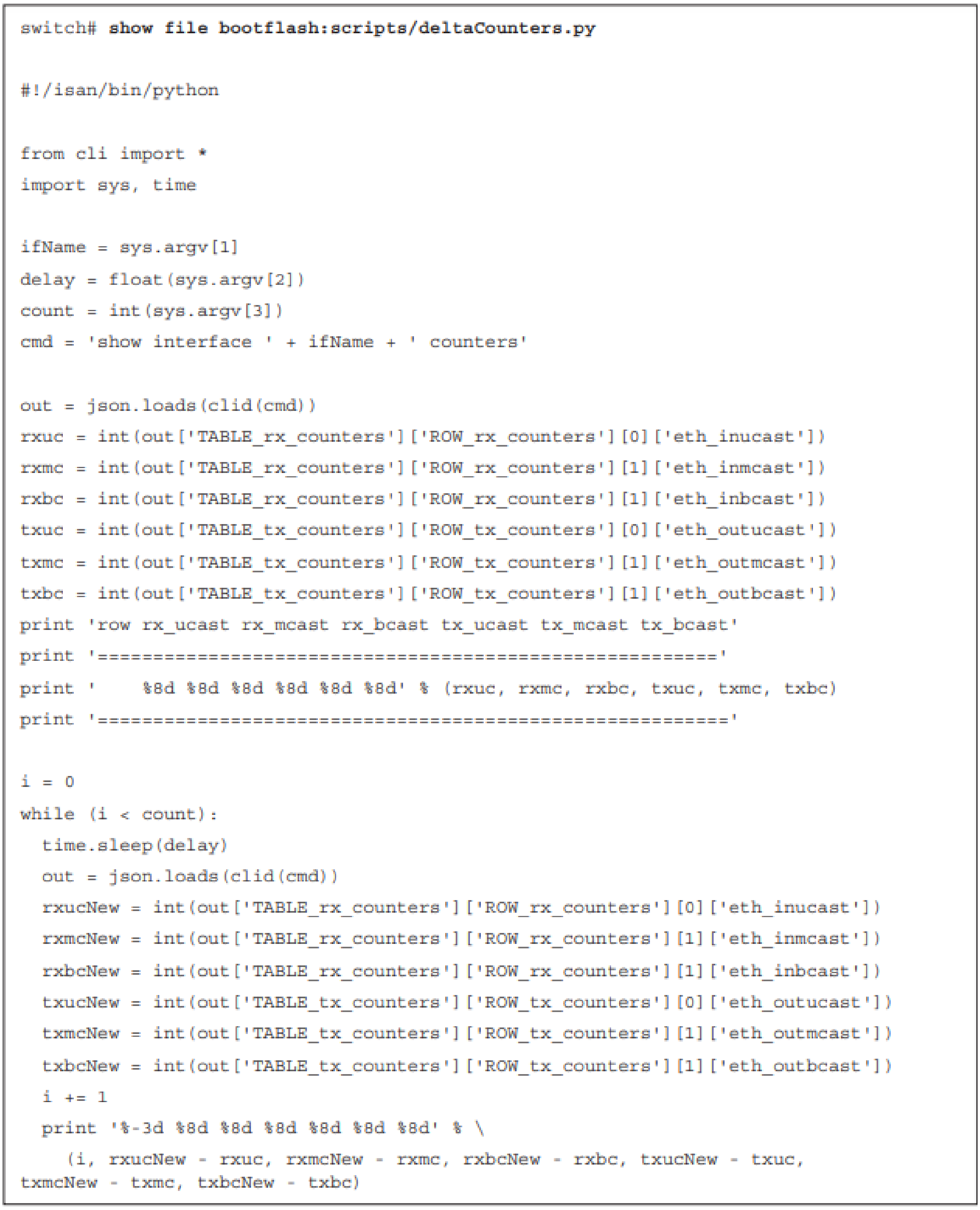
You can run a Python script in noninteractive mode by providing the Python script name as
an argument to the Python CLI command. Python scripts must be placed under the bootflash or volatile scheme. A maximum of 32 command-line arguments for the Python script
are allowed with the Python CLI command.
To execute a Python script, enter the python command, followed by the filename of the
script, followed by any arguments for the script, as shown in this example:

```
switch# python bootflash:scripts/deltaCounters.py Ethernet1/1
```

The Cisco Nexus switches also support the source CLI command for running Python scripts. The bootflash:scripts directory is the default script directory for the source CLI command.

```
switch# source deltaCounters Ethernet1/1 1 5
```

You can display the script source using the **show file CLI command**, as in Example below



```
switch# show file bootflash:scripts/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)
```

UCS Manager Python SDK

The Cisco UCS Python SDK is a Python module that helps automate all aspects of Cisco
UCS management, including server, network, storage, and hypervisor management.

The bulk of the Cisco UCS Python SDK works on the UCS Manager's Management Information Tree (MIT), performing create, modify, or delete actions on the managed objects (MO)
in the tree.

For login and logout from the UCS Manager, you need to import the UCSHandle class. The
following example shows how to create a connection handle before you can log in and log
out from the server.

```
from ucsmsdk.ucshandle import UCSHandle
Create a connection handle
handle = UcsHandle("192.168.1.1", "admin", "password")
Login to the server
handle.login()
Logout from the server
handle.logout()
```

The SDK provides APIs to enable CRUD operations:

- **Create an object:** add_mo
- **Retrieve an object:** query_dn, query_classid, query_dns, query_classids
- **Update an object:** set_mo
- **Delete an object:** delete_mo

Convert to UCS Python

Wouldn't it be cool if you didn't have to know much about the SDK to be able to automate
operations based off it?
Welcome the convert_to_ucs_python API!

The steps involved to generate a Python script equivalent to the steps performed on the
UCSM GUI are as follows:
Step 1. Launch the Java-based UCSM user interface (UI).

Step 2. Launch the Python shell and invoke convert_to_ucs_python on the same machine.
Step 3. Perform the desired operation on the UI.
Step 4. The convert_to_ucs_python API monitors the operation and generates equivalent Python script for it.
The UCSM GUI logs all the activities that are performed through it, and the Python shell
monitors that log to generate the equivalent Python script. Because the logging is local to
the machine where the UI is running, convert_to_ucs_python also must run on the same
machine.

PowerOn Auto Provisioning (POAP)

PowerOn Auto Provisioning (POAP) **automates the process of upgrading software images
and installing configuration files on devices that are being deployed in the network for the
first time**
.

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

Limitations of POAP

The switch software image must support POAP for this feature to function. POAP does not support provisioning of the switch after it has been configured and is operational. Only autoprovisioning of a switch with no startup configuration is supported.

Network Requirements for POAP

POAP requires the following network infrastructure, as shown in Figure

■ A DHCP server to bootstrap the interface IP address, gateway address, and Domain
Name System (DNS) server
■ A TFTP server that contains the configuration script used to automate the software
image installation and configuration process
■ One or more servers that contain the desired software images and configuration files

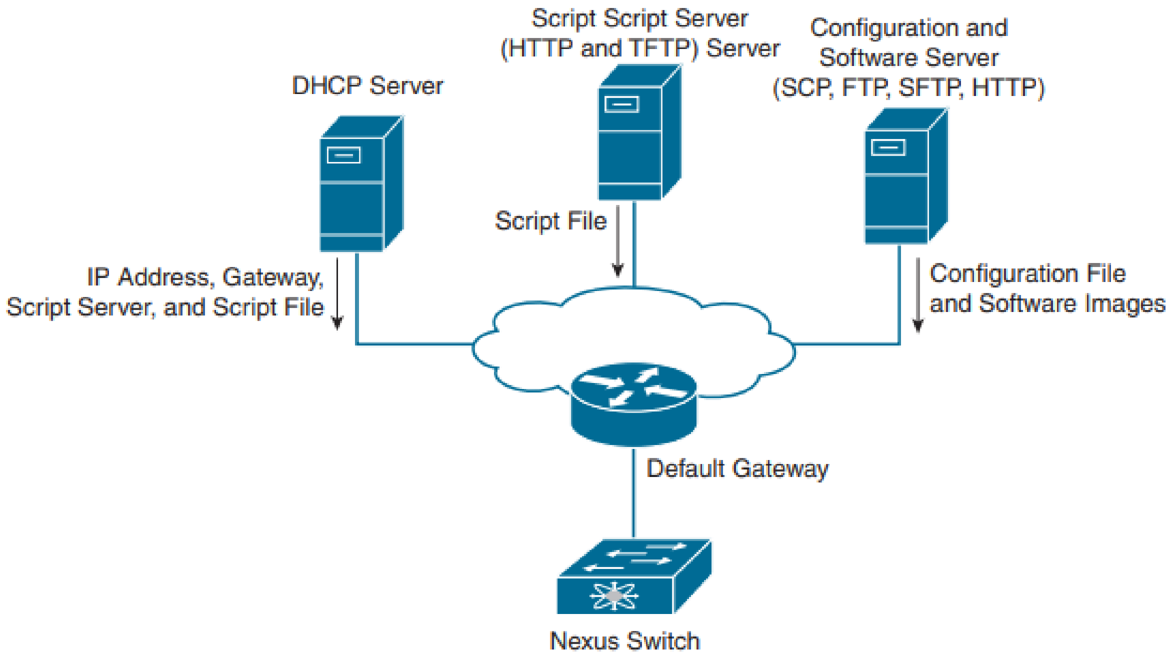


Figure 16-5 *POAP Network Infrastructure*

NOTE: Checking for a USB device containing the configuration script file in POAP mode is
not supported on Cisco Nexus 9000 Series switches

POAP Configuration Script

Cisco has sample configuration scripts that were developed using the Python programming
language and Tool command language (Tcl). You can customize one of these scripts to meet
the requirements of your network environment.

The reference script supplied by Cisco supports the following functionality:
■ Retrieves the switch-specific identifier—for example, the serial number.
■ Downloads the software image (system and kickstart images) if the files do not already exist
on the switch. The software image is installed on the switch and is used at the next reboot.
■ Schedules the downloaded configuration to be applied at the next switch reboot.
■ Stores the configuration as the startup configuration.

For Cisco Nexus 9000 Series switches, the POAP script can be found at https://github.com/
datacenter/nexus9000/blob/master/nx-os/poap/poap.py.

POAP Process

The POAP process has the following phases:

1. Power up
2. USB discovery
3. DHCP discovery
4. Script execution
5. Post-installation reload
   Within these phases, other process and decision points occur. Figure below shows a flow diagram of the POAP process.

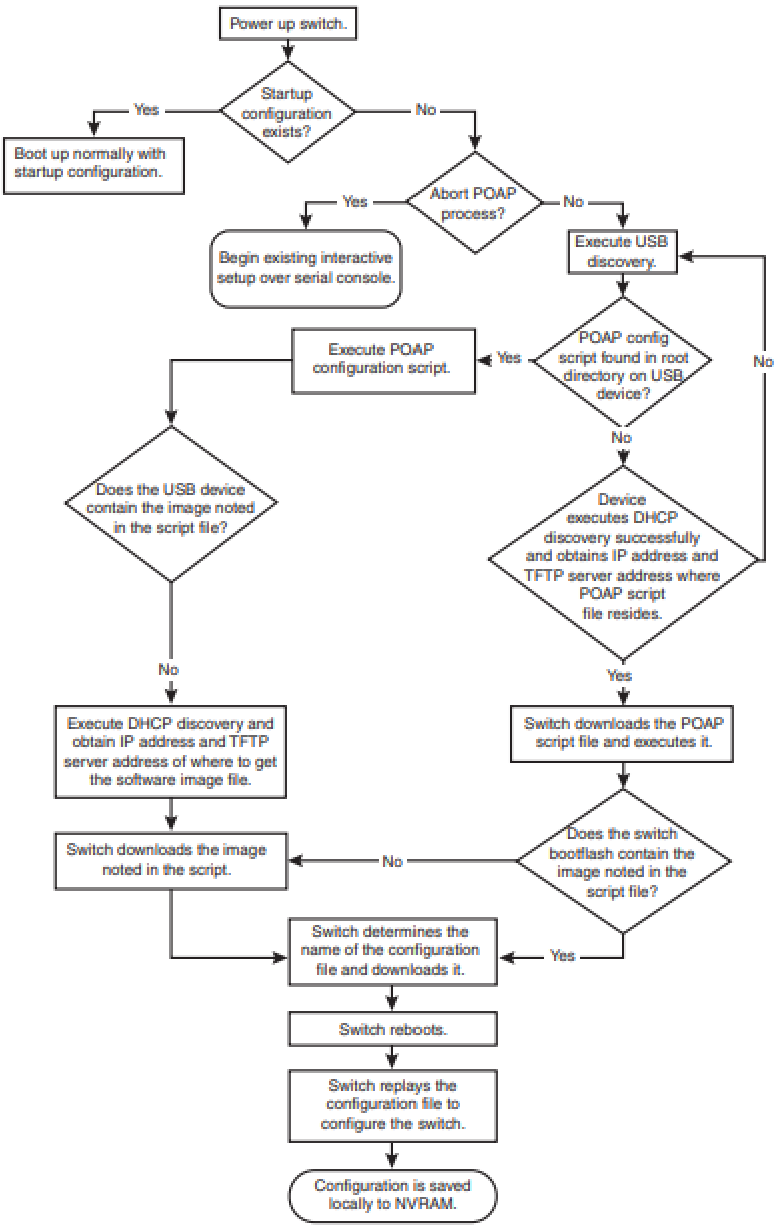

Power up switch.
Startup configuration exists?
Yes
No
Boot up normally with startup configuration.
Abort POAP process?
Yes
No
Begin existing interactive setup over serial console.
Execute USB discovery.
Execute POAP configuration script.
Yes
POAP config script found in root directory on USB device?
No
No
Does the USB device contain the image noted in the script file?
Device executes DHCP discovery successfully and obtains IP address and TFTP server address where POAP script file resides.
No
Yes
Execute DHCP discovery and obtain IP address and TFTP server address of where to get the software image file.
Switch downloads the POAP script file and executes it.
Switch downloads the image noted in the script.
No
Does the switch bootflash contain the image noted in the script file?
Switch determines the name of the configuration file and downloads it.
Yes
Switch reboots.
Switch replays the configuration file to configure the switch.
Configuration is saved locally to NVRAM.

Figure 16-6 *POAP Process*

Power-Up Phase

When you power up the device for the first time, it loads the software image that is installed
at manufacturing and tries to find a configuration file from which to boot. When a configuration file is not found, the POAP mode starts

**During startup, a prompt appears asking if you want to abort POAP and continue with a
normal setup. You can choose to exit or continue with POAP**

If you exit POAP mode, you enter the normal interactive setup script. If you continue in
POAP mode, all the front-panel interfaces are set up in the default configuration.

NOTE: No user intervention is required for POAP to continue. The prompt that asks if you
want to abort POAP remains available until the POAP process is complete.

USB Discovery Phase

When POAP starts, the process searches the root directory of all accessible USB devices for
the POAP configuration script file, configuration files, and system and kickstart images. If the
configuration script file is found on a USB device, POAP begins running the configuration
script. If the configuration script file is not found on the USB device, POAP executes DHCP
discovery.

If the software image and switch configuration files specified in the configuration script are
present, POAP uses those files to install the software and configure the switch.
If the software image and switch configuration files are not on the USB device, POAP starts the DHCP
phase from the beginning.

DHCP Discovery Phase

The switch sends out DHCP discover messages on the front-panel interfaces or the MGMT
interface that solicits DHCP offers from the DHCP server or servers (see Figure 16-7). The
DHCP client on the Cisco Nexus switch uses the switch serial number in the client-identifier
option to identify itself to the DHCP server. The DHCP server can use this identifier to send
information, such as the IP address and script filename, back to the DHCP client.

POAP requires a minimum DHCP lease period of 3600 seconds (1 hour). POAP checks the DHCP lease period. If the DHCP lease period is set to less than 3600 seconds (1 hour), POAP does not complete the DHCP negotiation.

The DHCP discover message also solicits the following options from the DHCP server:

TFTP server name or TFTP server address: The DHCP server relays the TFTP server name or TFTP server address to the DHCP client. The DHCP client uses this information to contact the TFTP server to obtain the script file

- **Bootfile name:** The DHCP server relays the bootfile name to the DHCP client. The bootfile name includes the complete path to the bootfile on the TFTP server. The DHCP client uses this information to download the script file.

the image below shows a flow diagram of the DHCP Discovery process.

Script Execution Phase

After the device bootstraps itself using the information in the DHCP acknowledgment, the script file is downloaded from the TFTP server

The switch runs the configuration script, which downloads and installs the software image and downloads a switch-specific configuration file.

However, the configuration file is not applied to the switch at this point, because the software image that currently runs on the switch might not support all of the commands in the configuration file. After the switch reboots, it begins running the new software image, if an image was installed. At that point, the configuration is applied to the switch.

NOTE: If the switch loses connectivity, the script stops, and the switch reloads its original

software images and bootup variables.

Post-Installation Reload Phase

The switch restarts and applies (replays) the configuration on the upgraded software image.
Afterward, the switch copies the running configuration to the startup configuration.

Configuring a Switch Using POAP

set up
to use POAP. The procedure to configure a switch using POAP is as follows:

Step 1. Install the switch in the network.

Step 2. Power on the switch. If no configuration file is found, the switch boots in POAP mode and displays a prompt asking if you want to abort POAP and continue with a normal setup. No entry is required to continue to boot in POAP mode.

Step 3. (Optional) If you want to exit POAP mode and enter the normal interactive setup script, enter y (yes).

Step 4. The switch boots, and the POAP process begins.

Step 5. Verify the device configuration using the commands shown in Table

Cisco DCNM

The Data Center Network Manager (DCNM) is the network management platform for all
NX-OS-enabled deployments, spanning new fabric architectures, IP Fabric for Media, and
storage networking deployments for the Cisco Nexus-powered data center. DCNM provides
management, control, automation, monitoring, visualization, and

troubleshooting across
Cisco Nexus and Cisco Multilayer Distributed Switching (MDS) solutions.

Cisco DCNM has the following benefits:

■ **Automation:** Accelerates provisioning from days to minutes and simplifies deployments

■ **Visibility:** Reduces troubleshooting cycles with graphical operational visibility for
topology, network fabric, and infrastructure

■ **Consistency:** Eliminates configuration errors with templated deployment models and
configuration compliance alerting with automatic remediation

DCNM can be deployed in four different variants, as illustrated below:

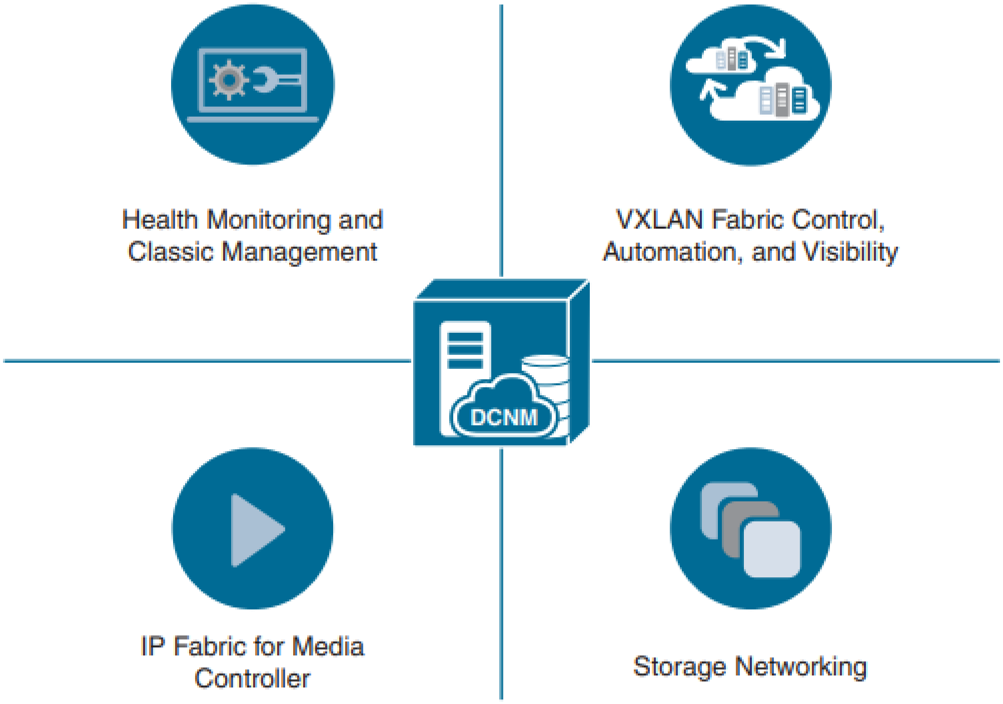

Health Monitoring and Classic Management
VXLAN Fabric Control, Automation, and Visibility
DCNM
IP Fabric for Media Controller
Storage Networking

Figure 16-8 *Data Center Network Manager (DCNM)*

■ **Classic LAN:** For managing classic LAN deployments such as VLANs and vPCs

■ **LAN Fabric:** For fabric management for multiple types of LAN solutions, including

VXLAN-EVPN, Cisco FabricPath, and traditional three-tier LAN deployments

- **Media Controller:** For managing media networks

- **SAN Management:** For managing SAN fabrics

Feature Details and Benefits

Cisco DCNM provides a robust framework and comprehensive feature set that meets the
routing, switching, and storage administration needs of present and future virtualized data
centers.

DCNM provides the following features for LAN Fabric with VXLAN-EVPN:

- Fabric control and overlay visibility
- Fabric Builder with PowerOn Auto Provisioning (POAP) infrastructure
- Fabric and VXLAN compliance management
- VXLAN overlay management
- Global fabric interface manager for VXLAN fabrics
- Top views and control
- Unified topology views
- Multisite manager search, monitoring
- Multifabric support
- Virtual machine and Virtual Routing and Forwarding (VRF) table search
- Per-fabric pool management
- Role-based access control (RBAC) for fabric objects

DCNM provides the following features for storage networking:

- Storage topology and visibility
- Telemetry and monitoring
- Zoning
- Advanced analysis
- Storage integration

DCNM provides the following features for IP Fabric for Media (IPFM):

- Flow control
- Visualization and health
- Provisioning and automation

DCNM provides the following features for automation and REST APIs:

- REST APIs
- REST and JavaScript Object Notation (JSON) API
- Multi-orchestrator support
- Automated discovery
- Provisioning GUI, tools, and wizard
- Customizable templates
- Configuration and change management
- Software image management

DCNM provides the following features for visibility, monitoring, and troubleshooting:

- Dashboards
- Topology views
- Topology overlays
- Performance and capacity management
- Health check and correction
- Host tracking
- VMware visibility
- Event management and alarms
- Reports

DCNM provides the following features for operations:

- Embedded database for enterprise deployments
- High availability deployment
- Event handling/forwarding

Cisco DCNM Web User Interface

Cisco DCNM provides a high level of visibility and control through a single web-based
management console.

A DCNM Classic LAN deployment has the following menus:

- Dashboard
- Topology
- Inventory
- Monitor
- Configure
- Administration
- Applications

A DCNM LAN Fabric deployment has the following menus:

- Dashboard
- Topology
- Control
- Monitor
- Administration
- Applications

A DCNM Media Controller deployment has the following menus:

- Dashboard
- Inventory
- Monitor
- Configure
- Media Controller
- Administration
- Catalog

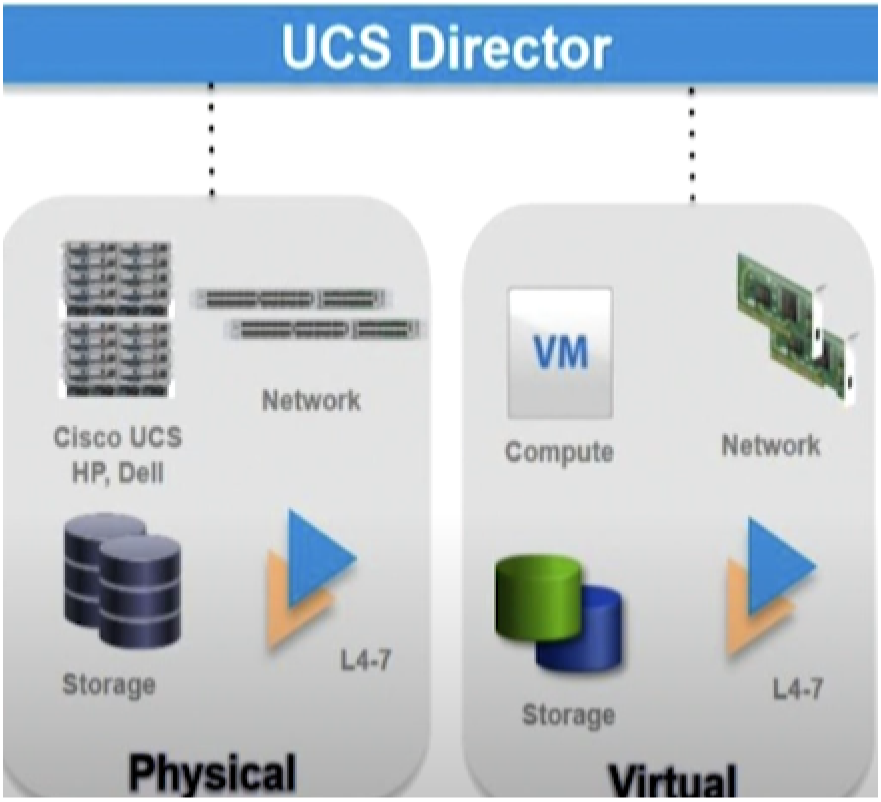
A DCNM SAN Management deployment has the following menus:

- Dashboard
- Topology
- Inventory
- Monitor

- Configure
- Administration

Cisco UCS Director

- Cisco Ucs Director is a workflow orchesteration tool that helps build IP services rapidelly
- Crete workflows to automate simplle or complex provisioning process for storage ,network,compute,and virtualization
- UCSD has over 1800 task library to orchestrate workflows
- Drag-n-Drop workflow creation



UCSD Use-Cases

- VM provisioning and lifcycle management
- Storage area network(SAN) Orchestration and lifecycle management
- Network resources provisioning and lifecycle management
- Application infrastructure provisioning -compute management
- VM provisioning

- Server Provisioning (Bare Metal)
- Tenant onboarding

Network Policy

- Network prolicy defines netwaork resources / rules (conditions)
- Which cloud provisioned VMs should be part of
- Minimum network requirements to be met
- Network port group name / type
- DHCP or static IP configuration while provisioning new VMs

Storage Policy

- Storage policy defines storage resources / rules (conditions)
- Data stores scope (ALL, include, exlude)
- Storage options (Local. SAN, NFS)
- Minimum conditgions on storage
- Deployment options
- Allow resizing of disk

Compute Policy

- Compite policy defines computing resources / rules
- Host node scope (include, exclude)
- Resorce pool
- ESX Type (ESX, ESXi or Any)
- ESX Version
- Minimum conditions
- Deployment options (VCPU...)
- Resizing Options
- Deploy folder