Cisco MDS 9000 Series Programmability Guide, Release 8.x

First Published: 2017-04-01

Last Modified: 2020-08-10

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

CONTENTS

PREFACE

Preface v

Preface v

Audience v

Document Conventions v

Related Documentation vi

Obtaining Documentation and Submitting a Service Request vii

CHAPTER 1

New and Changed Information 1

New and Changed Information 1

CHAPTER 2

NX-API 3

About NX-API 3

NX-API Workflow 4

NX-API Performance 4

About NX-API Messages 5

Message Format 5

Security 6

Limitations 6

Structured Output 6

About JSON 7

Configuring NX-API CLI 7

Sample NX-API Scripts 11

Examples of Structured Output 11

NX-API Developer Sandbox 15

NX-API Request Elements 26

NX-API Response Elements 29

Table of NX-API Response Codes 30
Default Settings 31
Additional References 32

CHAPTER 3 Python API 33
About the Python API 33
Supported Switches 34
Using Python 34
Cisco Python Package 34
Using the CLI Command APIs 35
Invoking the Python Interpreter from the CLI 36
Display Formats 37
Non-interactive Python 38
Running Scripts with Embedded Event Manager 39
Python Integration with Cisco MDS NX-OS Network Interfaces 40
Cisco MDS NX-OS Security with Python 40
Examples of Security and User Authority 40
Example of Running Script with Scheduler 42

CHAPTER 4 Ansible 43
Getting Started 43
Host File 43
Documentation 44
Example Playbook 44

CHAPTER 5 Cisco MDS SDK 47

Preface

This preface includes the following sections:

- Preface, on page v

Preface

This preface describes the audience, organization of, and conventions used in the Cisco MDS 9000 Series Configuration Guides. It also provides information on how to obtain related documentation, and contains the following chapters:

Audience

This publication is for experienced network administrators who configure and maintain Cisco Multilayer Director Switches (MDS) Devices.

Document Conventions

Command descriptions use the following conventions:

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

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

Examples use the following conventions:

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

This document uses the following conventions:

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Related Documentation

The documentation set for the Cisco MDS 9000 Series Switches includes the following documents.

Release Notes

http://www.cisco.com/c/en/us/support/storage-networking/mds-9000-nx-os-san-os-software/products-release-notes-list.html

Regulatory Compliance and Safety Information

http://www.cisco.com/c/en/us/td/docs/switches/datacenter/mds9000/hw/regulatory/compliance/RCSI.html

Compatibility Information

http://www.cisco.com/c/en/us/support/storage-networking/mds-9000-nx-os-san-os-software/products-device-support-tables-list.html

Installation and Upgrade

http://www.cisco.com/c/en/us/support/storage-networking/mds-9000-nx-os-san-os-software/products-installation-guides-list.html

Configuration

http://www.cisco.com/c/en/us/support/storage-networking/mds-9000-nx-os-san-os-software/products-installation-and-configuration-guides-list.html

CLI

http://www.cisco.com/c/en/us/support/storage-networking/mds-9000-nx-os-san-os-software/products-command-reference-list.html

Troubleshooting and Reference

http://www.cisco.com/c/en/us/support/storage-networking/mds-9000-nx-os-san-os-software/tsd-products-support-troubleshoot-and-alerts.html

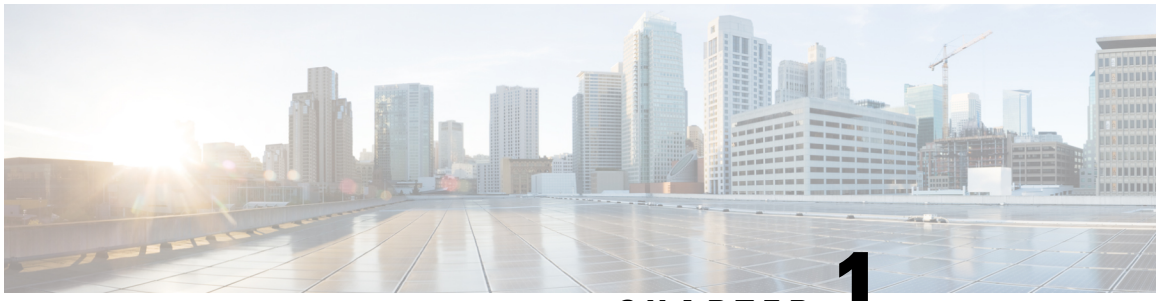
To find a document online, use the Cisco MDS NX-OS Documentation Locator at:

http://www.cisco.com/c/en/us/td/docs/storage/san_switches/mds9000/roadmaps/doclocater.html

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation*, at: http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html.

Subscribe to *What's New in Cisco Product Documentation*, which lists all new and revised Cisco technical documentation as an RSS feed and delivers content directly to your desktop using a reader application. The RSS feeds are a free service.



CHAPTER 1

New and Changed Information

• New and Changed Information, on page 1

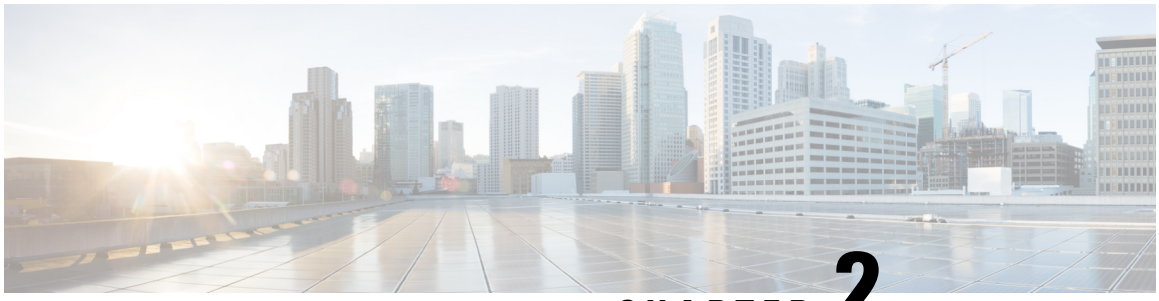
New and Changed Information

This table summarizes the new and changed features in the *Cisco MDS 9000 Series Programmability Guide*, and the sections in which they are documented.

Table 1: New and Changed Features

New/Enhanced Features	Description	Introduced in Cisco MDS NX-OS Release	Topic Where Documented
NX-API	Support for installing NX-API certificates with encrypted private key and using trust point certificate.	8.5(1)	NX-API, on page 3
Python 3.0	Support for Python 3.0 is added.	8.4(2)	Python API, on page 33
Ansible	Support for Ansible is added.	8.4(1)	Ansible, on page 43
NX-API	The **cli_show_array** command type support is added. The NX-API Developer Sandbox was modified. The **Command Reference** option is added. Support for Java and JavaScript code format is added.	8.4(1)	NX-API, on page 3

New/Enhanced Features	Description	Introduced in Cisco MDS NX-OS Release	Topic Where Documented
NX-API	The NX-API over HTTPS self-signed certificate expiry time is modified in the NX-OS 8.3(1) release.	8.3(1)	NX-API, on page 3
NX-API	NX-API is an enhancement to the Cisco MDS 9000 Series CLI system. Cisco MDS 9000 NX-API is an RPC-style API, taking and executing CLI commands. Based on HTTP or HTTPS protocols as common to other Representational State Transfer (REST) API frameworks, it allows programmatic access to a Cisco MDS switch. NX-API provides the configuration and management capabilities of Cisco MDS NX-OS CLI with a modern web-based API, enabling users to control a Cisco MDS switch using a web browser. When coupled with a programming language like Python and the appropriate libraries, it facilitates storage networking automation.	7.3(0)D1(1)	NX-API, on page 3



CHAPTER 2

NX-API

This chapter contains the following sections:

- About NX-API, on page 3
- NX-API Workflow, on page 4
- NX-API Performance, on page 4
- About NX-API Messages, on page 5
- Message Format, on page 5
- Security, on page 6
- Limitations, on page 6
- Structured Output, on page 6
- Configuring NX-API CLI, on page 7
- Sample NX-API Scripts, on page 11
- Examples of Structured Output , on page 11
- NX-API Developer Sandbox, on page 15
- NX-API Request Elements, on page 26
- NX-API Response Elements, on page 29
- Default Settings, on page 31
- Additional References, on page 32

About NX-API

NX-API is an enhancement to the Cisco MDS 9000 Series CLI system.

Cisco MDS 9000 NX-API is an RPC-style API, taking and executing CLI commands. Based on HTTP or HTTPS protocols as common to other Representational State Transfer (REST) API frameworks, it allows programmatic access to a Cisco MDS switch. NX-API provides the configuration and management capabilities of Cisco MDS NX-OS CLI with a modern web-based API, enabling users to control a Cisco MDS switch using a web browser. When coupled with a programming language like Python and the appropriate libraries, it facilitates storage networking automation.

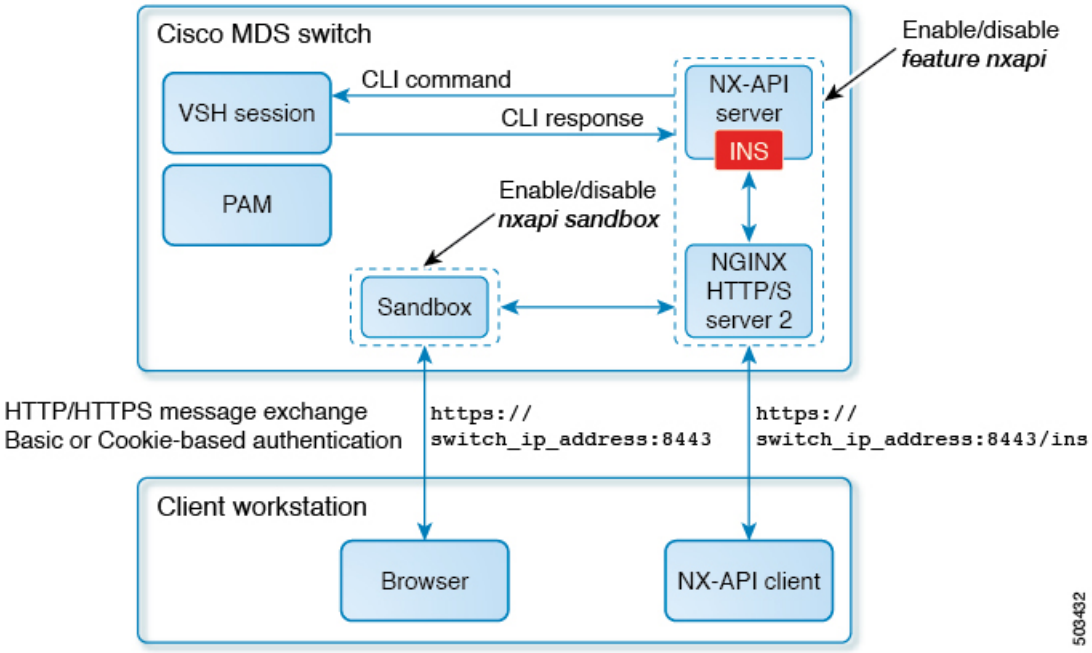
Cisco MDS NX-API supports certain **show** commands and configuration commands that are noninteractive.

Note A noninteractive command is a command that does not prompt the user to enter an input from the keyboard to proceed further.

NX-API Workflow

The NX-API backend uses the NGINX HTTP server. The NGINX server interfaces between the external client and the NXAPI server in the switch.

Figure 1: NX-API Workflow



NX-API Performance

NX-API throughput performance depends on the following factors:

- HTTP and HTTPS—NX-API performance on an HTTP server is better compared to that on an HTTPS server. This is because an HTTPS server has an overhead of encrypting and decrypting data to provide more security.
- Cisco MDS Switches (memory and process limitation)—NX-API performance is better in devices with more memory.
- Command output size—NX-API performance is better when the command outputs are smaller.
- Structured and unstructured outputs of the **show** commands—NX-API performance is better with unstructured outputs. Commands that support structured outputs are also called as **NX-API aware** commands in the document.

About NX-API Messages

HTTP Header

A header allows a client and a server to pass extra information as colon separated property-value pairs in requests and responses.

This is where the content encoding is specified for an NX-API request. The supported content types are:

Tag	Description	Type	Values
content-type	Request encoding type	string	application/json application/json-rpc application/xml

HTTP Method

Cisco MDS NX-API uses the POST method.

Message Body

The message body or payload contains the data for the HTTP method. For a list of supported objects see the NX-API Request Elements section.

Message Response

The message response is an HTTP return code and an HTTP response body that contains the data returned by the method. For a list of supported elements see the NX-API Respone Elements section. For a list of response codes refer to the NX-API Response Codes section.

Message Format

- Cisco NX-API output of supported commands can be viewed in XML, JSON, and JSON-RPC. This message format can be used for both requests and responses.
  - XML—Cisco NX-API proprietary protocol for delivering Cisco MDS NX-OS CLI commands in an XML payload.
  - JSON—Cisco NX-API proprietary protocol for delivering Cisco MDS NX-OS CLI commands in a JSON payload.
  - JSON-RPC—A standard lightweight remote procedure call (RPC) protocol that can be used to deliver Cisco MDS NX-OS CLI commands in a JSON payload. The JSON-RPC 2.0 specification is outlined by jsonrpc.org.

NX-API does not map directly to the Cisco NX-OS NETCONF implementation.

Security

By default, Cisco MDS NX-API uses HTTP basic authentication (that is, all command requests must contain the username and password of the device in the HTTP header). NX-API can also leverage HTTPS to secure and encrypt data. An HTTPS connection provides more security over an HTTP connection. NX-API provides session-based cookie authentication as an alternative to the HTTP authentication method.

On Cisco NX-OS Releases 8.1(x) and 8.2(x), when NX-API is enabled over HTTPS, a 2048-bit SHA-1 self-signed certificate is created. This certificate is valid for two years. When an expired certificate is used, the browser displays a warning about security vulnerabilities. To avoid such vulnerabilities, we recommend the use of a CA-signed certificate. From Cisco NX-OS Release 8.3(1) and later, the self-signed certificate expires after 24 hours. We recommend that you use a CA-signed certificate.

For information on configuring CA-signed certificates, see the Configuring Certificate Authorities and Digital Certificates section in the *Cisco MDS 9000 Series Security Configuration Guide, Release 8.x*.

NX-API is integrated into the CLI authentication system of the Cisco MDS switch. This means that users must have the appropriate privilege to run CLI commands on the switch that are posted through NX-API. For example, a user with read only privileges on a Cisco MDS 9000 switch, cannot execute configuration commands through NX API.

NX-API performs authentication through a programmable authentication module (PAM) on a switch. Use cookies to reduce the number of PAM authentications, which in turn reduces the load on the PAM.

NX-API provides a session-based cookie, nxapi_auth when users first authenticate successfully. An nxapi_auth cookie expires in 600 seconds (10 minutes). This value is fixed and cannot be configured. The session cookie is used to avoid reauthentication during communication. If the session-based cookie is not included with subsequent requests, another session-based cookie is required; this is obtained through a full authentication process. Avoiding unnecessary use of the authentication process helps to reduce the workload of the MDS switch.

Limitations

- The XML output for FCIP interface related commands is not supported.
- The XML output for consistency checker commands is not supported.

Structured Output

The NX-OS supports redirecting the standard output of various **show** commands in the following structured output formats:

- XML
- JSON. The limit for JSON output is 60 MB.
- JSON Native

Converting the standard Cisco MDS NX-OS output to any of these formats occurs on the Cisco MDS NX-OS CLI by "piping" the output to a JSON, JSON Native, or a XML interpreter. The JSON and XML interpreters

are built-in into Cisco MDS NX OS software. For example, you can issue the **show ip access** command with the logical pipe (|) and specify the output format.The Cisco MDS NX-OS command output is properly structured and encoded in that format. This feature enables programmatic parsing of the data and supports streaming data from the switch through software streaming telemetry.

For more information on how to select different output formats, see NX-API Developer Sandbox, on page 15 section.

From Cisco MDS NX-OS Release 8.3(1), Cisco has implemented an enhanced version of JSON called JSON Native. This is a new CLI option to choose from. JSON Native displays the JSON output faster and more efficiently by bypassing an extra layer of command interpretation. In fact, JSON Native preserves the data type in the output; it displays integers as integers instead of converting them to a string for an output. We recommend to use JSON Native.

About JSON

JavaScript Object Notation (JSON) is a light-weight text-based open standard that is designed for human-readable data and is an alternative to XML. JSON was originally designed from JavaScript, but it is language-independent data format. JSON and JSON Native are supported for command output.

The two primary data structures that are supported in some way by all modern programming languages are as follows:

- Ordered List of values—Often known as Array or List (for example, it is List in Python)
- Collection of Key/Value pairs—Often known as Objects or Dictionary (for example, it is Dictionary in Python)

CLI Execution

```
switch-1-vxlan-1# show cdp neighbors  | json

{
  "TABLE_cdp_neighbor_brief_info": {
    "ROW_cdp_neighbor_brief_info": {
      "ifindex": 83886080,
      "device_id": "SW-SPARSHA-II17-PUB",
      "intf_id": "mgmt0",
      "ttl": 146,
      "capability": [
        "switch",
        "IGMP_cnd_filtering"
      ],
      "platform_id": "cisco WS-C2960S-48TD-L",
      "port_id": "GigabitEthernet1/0/1"
    }
  },
  "neigh_count": 1
}
```

Configuring NX-API CLI

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

For more details about NX-API response codes, see Table of NX-API Response Codes, on page 30.

After configuring NX-API on the MDS switch, it may be accessed through the following URLs:

- HTTP - http://*switch_ip_address:port-number*/ins
- HTTPs - https://*switch_ip_address:port-number*/ins

For default HTTP and HTTPS settings, refer the Default Settings, on page 31 section.

The following example shows how to configure and enable NX-API:

1. Ensure that the switch is accessible through the management interface.

   Refer to the Configuring the Management Interface section in the *Cisco MDS 9000 Series Fundamentals Configuration Guide* on how to enable the management interface.

2. Enable the NX-API feature.

```
switch# configure terminal
switch(config)# feature nxapi
```

3. (Optional) Disable the NX-API feature.

```
switch(config)# no feature nxapi
```

4. After configuring NX-API on the MDS switch, it may be accessed through the HTTP/HTTPS ports:

   (Optional) Configure HTTP port for NX-API.

```
switch(config)# nxapi http port 8080
```

   Use the **no** form of the command to disable it.

   (Optional) Configure HTTPS port for NX-API.

```
switch(config)# nxapi https port 8443
```

   Use the **no** form of the command to disable it.

5. (Optional) Install an identity certificate for NX-API HTTPS connections. Either a trust point or NX-API certificate may be used. You cannot configure both sources at the same time.

   a. Install a certificate with an unencrypted private key that is used only by the NX-API feature:

```
switch(config)# nxapi certificate certfile  key keyfile
```

   b. Install a certificate with an encrypted private key that is used only by the NX-API feature:

```
switch(config)# nxapi certificate certfile  key keyfile  password  passphrase
```

   c. Use an already installed certificate in the trustpoint repository that may be shared with other features:

```
switch(config)# nxapi trustpoint label
```

Note Installing a new NX-API certificate will reset the NX-API server. Installing a certificate from a host with NX-API may cause the script to fail.

For information about configuring trust points, see the Configuring Certificate Authorities and Digital Certificates chapter in the Cisco MDS 9000 Series Security Configuration Guide, Release 8.x.

- *certfile* is a signed certificate for this switch in privacy-enhanced mail (PEM) format. PEM format is a standard file format for storing and sending cryptographic RSA keys, certificates, and other data, based on a set of 1993 IETF standards.
- *keyfile* is the private key for this switch in the PEM format. If the key is encrypted then the **password** option must also be specified.
- *passphrase* is the password that is used to encrypt the private key.
- *label* is the name of an already configured cryptographic trust point.

Note

- Certificates and keys installed using the **nxapi certificate key** command are not shared with any other crytographic features on the switch.
- The **password** option used in the **nxapi certificate key** command is available only from Cisco MDS NX-OS Release 8.5(1).

6. (Optional) If required, allow weak SSL ciphers for NX-API HTTPS connections. This reduces the security of SSL connections. However, this may be required for older devices to communicate with the switch.

```
switch(config)# nxapi ssl ciphers weak
```

7. (Optional) If required, configure SSL transports for NX-API HTTPS connections. Enabling nondefault older transports reduces the security of SSL connections. However, this may be required for older devices to communicate with the switch.

```
switch(config)# nxapi ssl protocols TLSv1.1 TLSv1.2
```

Preparing an Identity Certificate for Use in NX-API

An identity certificate for NX-API must be created before it can be imported with the **nxapi certificate** command. The certificate must consist of the switch identity certificate only; all CA and intermediate authority certificates must be removed. The private key must be removed and the total size must be less than 4096 bytes.

If a switch identity certificate is already installed in the switch crypto infrastructure under a trustpoint, this may be exported and reformattted, and the private key extracted, to be used in NX-API. If there is no switch identity certificate already installed, then it needs to be created by the CA.

For information on how to create a certificate, see the Configuring Certificate Authorities and Digital Certificates chapter in the *Cisco MDS 9000 Series Security Configuration Guide, Release 8.x.*

Note The tools to prepare an existing identity certificate for NX-API use are not available on the switch. This must be done on another device such as a host with OpenSSL installed.

1. (Optional) If a switch identity certificate is already installed on the switch, export it in PKCS12 format using the following command:

```
switch(config)# crypto ca export trustpoint_name pkcs12 mytpexport.pkcs12 my_passphrase
```

2. Upload the file to a host with OpenSSL installed on it:

```
switch# copy mytpexport.pkcs12 sftp://10.10.2.2
```

3. Extract the identity certificate:

```
host$ openssl pkcs12 -in mytpexport.pkcs12 -nokeys -clcerts -out idcert.pem
Enter Import Password: my_passphrase
host$
```

4. Extract unexcrypted private key:

```
host$ openssl pkcs12 -in mytpexport.pkcs12 -nocerts -nodes | openssl rsa -out
unencryptedprivkey.pem
Enter Import Password:
writing RSA key
host$
```

5. Download the 2 files to the switch bootflash:

```
switch# copy sftp://10.10.2.2/idcert.pem bootflash:
switch# copy sftp://10.10.2.2/unencryptedprivkey.pem bootflash:
```

The files are now ready to be imported using the **nxapi certificate** command.

Using NX-API with cURL

Let us now examine the content of the **show.version.json** file on the host.

```
linux$ cat show.version.json
[{ "jsonrpc": "2.0", "method": "cli", "params": { "cmd": "show version", "version": 1 },
"id": 1 }]
EOF
```

Now use cURL on host to authenticate the switch and send it the desired POST request.

```
linux$ curl -v -u admin:cisco -H "Content-Type: application/json-rpc" -H "Cache-Control:
no-cache" -d @show.version.json -X POST http://10.10.2.2:80/ins

Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 10.10.2.2:80...
* Connected to 10.10.2.2:80 (10.10.2.2:80) port 80 (#0)
* Server auth using Basic with user 'admin'
> POST /ins HTTP/1.1
> Host: 10.10.2.2:80
> Authorization: Basic YWRtaW46bmJ2XzEyMzQ1
> User-Agent: curl/7.70.0
> Accept: */*
> Content-Type: application/json-rpc
> Cache-Control: no-cache
> Content-Length: 99
>
* upload completely sent off: 99 out of 99 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.7.10
< Date: Mon, 14 Jun 1976 13:28:43 GMT
< Content-Type: application/json-rpc; charset=UTF-8
< Transfer-Encoding: chunked
< Connection: keep-alive
< Set-Cookie: nxapi_auth=dzqnf:1fNa+E8KGq0ZZM6TRZTFKTWejBg=; Secure; HttpOnly;
< X-Frame-Options: SAMEORIGIN
< X-XSS-Protection: 1; mode=block
< X-Content-Type-Options: nosniff
< Strict-Transport-Security: max-age=31536000; includeSubDomains
< Content-Security-Policy: block-all-mixed-content; base-uri 'self'; default-src 'self';
script-src 'self'; style-src 'self'; img-src 'self' ; connect-src 'self'; font-src 'self';
```

```
 object-src 'none'; media-src 'self'; form-action 'self'; frame-ancestors 'self';
<
{
        "jsonrpc":      "2.0",
        "result":       {
                "header_str":   "Cisco Nexus Operating System (NX-OS) Software\nTAC support:
 http://www.cisco.com/tac\nDocuments:
http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home.html\nCopyright
 (c) 2002-2020, Cisco Systems, Inc. All rights reserved.\nThe copyrights to certain works
contained herein are owned by\nother third parties and are used and distributed under
license.\nSome parts of this software are covered under the GNU Public\nLicense. A copy of
 the license is available at\nhttp://www.gnu.org/licenses/gpl.html.\n",
                "bios_ver_str": "2.1.17",
                "loader_ver_str":       "N/A",
                "kickstart_ver_str":    "8.4(1)SK(0) [build 8.4(1)SK(0.160)] [gdb]",
                "sys_ver_str":  "8.4(1)SK(0) [build 8.4(1)SK(0.160)] [gdb]",
                "bios_cmpl_time":       "01/08/14",
                "kick_file_name":       "bootflash:///kick-sky160",
                "kick_cmpl_time":       " 12/20/2020 12:00:00",
                "kick_tmstmp":  "09/08/2020 09:42:15",
                "isan_file_name":       "bootflash:///sky-sep14-02",
                "isan_cmpl_time":       " 12/20/2020 12:00:00",
                "isan_tmstmp":  "09/14/2020 05:56:35",
                "chassis_id":   "MDS 9250i 40 FC 2 IPS 8 FCoE (2 RU) Chassis",
                "module_id":    "40FC+8FCoE+2IPS Supervisor",
                "cpu_name":     "Motorola, e500v2",
                "memory":       4088480,
                "mem_type":     "kB",
                "proc_board_id":        "JAF1852AAFC",
                "host_name":    "host",
                "bootflash_size":       4001760,
                "kern_uptm_days":       0,
                "kern_uptm_hrs":        1,
                "kern_uptm_mins":       25,
                "kern_uptm_secs":       13,
                "rr_usecs":     715180,
                "rr_ctime":     "Mon Jun 14 12:02:47 1976",
                "rr_reason":    "Reset Requested by CLI command reload",
                "rr_sys_ver":   "8.4(1)SK(0.160)",
                "rr_service":   "",
                "manufacturer": "Cisco Systems, Inc."
        },
        "id":   1,
        "cmd":  "show version"
* Connection #0 to host 10.197.155.246 left intact
```

Sample NX-API Scripts

You can access sample scripts that demonstrate how to use a script with NX-API. To access a sample script, click the following link then choose the directory that corresponds to the required software release: https://github.com/datacenter/nxos/tree/master/nxapi/samples.

Examples of Structured Output

This section lists a selected few examples of Cisco MDS NX-OS commands that are displayed as XML, JSON and JSON Native output formats.

To check if a particular **show** command is NX-API-aware, enter the command along with | **xml** on the switch:

command | xml

If a command is NX-API-aware (supports structured output), the resulting output is in XML format:

```
switch# show device-alias merge status | xml

 <?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns="http://www.cisco.com/nxos:8.4.1.SK.0.:ddas"
xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">
<nf:data>
<show>
<device-alias>
<merge>
<status>
<__readonly__>
<result>Success</result>
<reason>None</reason>
</__readonly__>
</status>
</merge>
</device-alias>
</show>
</nf:data>
</nf:rpc-reply>
]]>]]>
```

If a command is not NX-API-aware, the resulting output has the following error:

```
switch# show logging logfile | xml

Error: This command does not support XML output.
```

This example shows how to display the **show version** command in the XML format:

```
switch(config)# show version | xml

<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns="http://www.cisco.com/nxos:8.4.2.:sysmgrcli"
xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0">
 <nf:data>
  <show>
   <version>
    <__readonly__>
     <header_str>Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Documents: http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home.html
Copyright (c) 2002-2020, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under
license. Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or the GNU
Lesser General Public License (LGPL) Version 2.1. A copy of each
such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://www.opensource.org/licenses/lgpl-2.1.php
</header_str>
     <bios_ver_str>3.7.0</bios_ver_str>
     <kickstart_ver_str>8.4(2) [build 8.4(2.191)] [gdb]</kickstart_ver_str>
     <sys_ver_str>8.4(2) [build 8.4(2.191)] [gdb]</sys_ver_str>
     <bios_cmpl_time>04/01/2019</bios_cmpl_time>
    <kick_file_name>bootflash:///m9700-sf3ek9-kickstart-mzg.8.4.2.191.bin</kick_file_name>

     <kick_cmpl_time> 2/5/2020 12:00:00</kick_cmpl_time>
```

```
     <kick_tmstmp>01/08/2020 18:27:03</kick_tmstmp>
     <isan_file_name>bootflash:///m9700-sf3ek9-mzg.8.4.2.191.bin</isan_file_name>
     <isan_cmpl_time> 2/5/2020 12:00:00</isan_cmpl_time>
     <isan_tmstmp>01/14/2020 05:36:15</isan_tmstmp>
     <chassis_id>MDS 9706 (6 Slot) Chassis</chassis_id>
     <module_id>Supervisor Module-3</module_id>
     <cpu_name>Intel(R) Xeon(R) CPU C5528 @ 2.13GHz</cpu_name>
     <memory>8167228</memory>
     <mem_type>kB</mem_type>
     <proc_board_id>JAE19220AQJ</proc_board_id>
     <host_name>abc</host_name>
     <bootflash_size>3915776</bootflash_size>
     <slot0_size>0</slot0_size>
     <kern_uptm_days>19</kern_uptm_days>
     <kern_uptm_hrs>23</kern_uptm_hrs>
     <kern_uptm_mins>16</kern_uptm_mins>
     <kern_uptm_secs>11</kern_uptm_secs>
     <rr_usecs>768558</rr_usecs>
     <rr_ctime>Tue Jan 14 05:58:26 2020</rr_ctime>
     <rr_reason>Reset Requested by CLI command reload</rr_reason>
     <rr_sys_ver>8.4(2.171)</rr_sys_ver>
     <rr_service></rr_service>
     <manufacturer>Cisco Systems, Inc.</manufacturer>
    </__readonly__>
   </version>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
```

This example shows how to display the **show version** in the JSON format:

```
switch(config)# show version | json

{
  "header_str": "Cisco Nexus Operating System (NX-OS) Software\nTAC support:
http://www.cisco.com/tac\nDocuments: http://www.cisco.c
om/en/US/products/ps9372/tsd_products_support_series_home.html\nCopyright (c) 2002-2020,
Cisco Systems, Inc. All rights reserved.\nT
he copyrights to certain works contained in this software are\nowned by other third parties
 and used and distributed under\nlicense.
 Certain components of this software are licensed under\nthe GNU General Public License
(GPL) version 2.0 or the GNU\nLesser General
 Public License (LGPL) Version 2.1. A copy of each\nsuch license is available
at\nhttp://www.opensource.org/licenses/gpl-2.0.php and
\nhttp://www.opensource.org/licenses/lgpl-2.1.php",
  "bios_ver_str": "3.7.0",
  "kickstart_ver_str": "8.4(2) [build 8.4(2.191)] [gdb]",
  "sys_ver_str": "8.4(2) [build 8.4(2.191)] [gdb]",
  "bios_cmpl_time": "04/01/2019",
  "kick_file_name": "bootflash:///m9700-sf3ek9-kickstart-mzg.8.4.2.191.bin",
  "kick_cmpl_time": "2/5/2020 12:00:00",
  "kick_tmstmp": "01/08/2020 18:27:03",
  "isan_file_name": "bootflash:///m9700-sf3ek9-mzg.8.4.2.191.bin",
  "isan_cmpl_time": "2/5/2020 12:00:00",
  "isan_tmstmp": "01/14/2020 05:36:15",
  "chassis_id": "MDS 9706 (6 Slot) Chassis",
  "module_id": "Supervisor Module-3",
  "cpu_name": "Intel(R) Xeon(R) CPU C5528 @ 2.13GHz",
  "memory": 8167228,
  "mem_type": "kB",
  "proc_board_id": "JAE19220AQJ",
  "host_name": "abc",
  "bootflash_size": 3915776,
```

```
  "slot0_size": 0,
  "kern_uptm_days": 19,
  "kern_uptm_hrs": 23,
  "kern_uptm_mins": 16,
  "kern_uptm_secs": 22,
  "rr_usecs": 768558,
  "rr_ctime": "Tue Jan 14 05:58:26 2020",
  "rr_reason": "Reset Requested by CLI command reload",
  "rr_sys_ver": "8.4(2.171)",
  "rr_service": null,
  "manufacturer": "Cisco Systems, Inc."
}
```

This example shows how to display the **show version** in the JSON Native format:

```
switch(config)# show version | json native

{
"header_str": "Cisco Nexus Operating System (NX-OS) Software\nTAC supp
ort: http://www.cisco.com/tac\nDocuments: http://www.cisco.com/en/US/products/ps
9372/tsd_products_support_series_home.html\nCopyright (c) 2002-2020, Cisco Syste
ms, Inc. All rights reserved.\nThe copyrights to certain works contained herein
are owned by\nother third parties and are used and distributed under license.\nS
ome parts of this software are covered under the GNU Public\nLicense. A copy of
the license is available at\nhttp://www.gnu.org/licenses/gpl.html.\n",
"bios_ver_str": "2.1.18",
"loader_ver_str": "N/A",
"kickstart_ver_str": "8.4(2a)",
"sys_ver_str": "8.4(2a)",
"bios_cmpl_time": "04/06/20",
"kick_file_name": "bootflash:///m9100-s5ek9-kickstart-mz.8.4.2a.bi
n",
"kick_cmpl_time": " 7/11/2020 12:00:00",
"kick_tmstmp": "06/20/2020 20:50:09",
"isan_file_name": "bootflash:///m9100-s5ek9-mz.8.4.2a.bin",
"isan_cmpl_time": " 7/11/2020 12:00:00",
"isan_tmstmp": "06/20/2020 22:05:47",
"chassis_id": "MDS 9148S 16G 48 FC (1 Slot) Chassis",
"module_id": "2/4/8/16 Gbps FC/Supervisor",
"cpu_name": "Motorola, e500v2",
"memory": 4088620,
"mem_type": "kB",
"proc_board_id": "JAF1751BGPS",
"host_name": "sw109-Mini",
"bootflash_size": 4001760,
"kern_uptm_days": 7,
"kern_uptm_hrs": 1,
"kern_uptm_mins": 13,
"kern_uptm_secs": 0,
"rr_usecs": 362070,
"rr_ctime": "Mon Sep 28 07:43:36 2020",
"rr_reason": "Reset due to upgrade",
"rr_sys_ver": "8.4(2b)",
"rr_service": "",
"manufacturer": "Cisco Systems, Inc."
}
```

NX-API Developer Sandbox

The NX-API Developer Sandbox is a Cisco-developed web-based user interface that is used to make NX-API requests and receive responses. Requests are in the form of **show** commands, and noninteractive configuration commands.

Configuring NX-API Sandbox

1. Enable and configure the NX-API feature. For information on how to enable and configure the NX-API feature, refer the Configuring NX-API CLI, on page 7 section.
2. Enable the NX-API sandbox:

```
switch# configure terminal
switch(config)# nxapi sandbox
```

To view the NX-API settings, use **show nxapi** command.

```
switch# show nxapi

NX-API:       Enabled         Sandbox:      Enabled
HTTP Port:    Disabled        HTTPS Port:  8443

Certificate Information:
    Issuer:   C=US, ST=CA, L=San Jose, O=Cisco Systems Inc., OU=dcnxos, CN=nxos
    Expires:  Nov 26 09:26:12 2019 GMT
    Content:  -----BEGIN CERTIFICATE-----
MIIDpzCCAo+gAwIBAgIJAObmdczeHJL8MA0GCSqGSIb3DQEBCwUAMGoxCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJDQTERMA8GA1UEBwwIU2FuIEpvc2UxGzAZBgNVBAoM
EkNpc2NvIFN5c3RlbXMgSW5jLjEPMA0GA1UECwwGZGNueG9zMQ0wCwYDVQQDDARu
eG9zMB4XDTE5MTEyNTA5MjYxMloXDTE5MTEyNjA5MjYxMlowajELMAkGA1UEBhMC
VVMxCzAJBgNVBAgMAkNBMREwDwYDVQQHDAhTYW4gSm9zZTEbMBkGA1UECgwSQ2lz
Y28gU3lzdGVtcyBJbmMuMQ8wDQYDVQQLDAZkY254b3MxDTALBgNVBAMMBG54b3Mw
ggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDreVT3LAYarHxZxELwNwst
ZQqlmah5PJHBGUx+3YQcRi8v8wrEsRI7bZrQgDzXkFEH9yroJUghdc0nkM1GYHNF
avbj4qRaEveRejtGZNMPi11tAOWbRsU0ldxODV3+SeG/A220Bal58JzJjje5wyi8
Wu8UR8w4Lb32GYXI8ifBtlU0LrNsS0pE68yJt782y8IJIAEjGWX3L8dle4qwRqXg
6GBDPbwKFCvi+NX8JN48olONASSHXGdcmZSfiYUNKPD7+AwjG/luxSyIqnFW2w06
zwQLoIbXJx7zvl0Adt3H4ZnaRZOG5UvsWLgEdJZJkaqQPe0+7cpwcubN9/PJlO0l
AgMBAAGjUDBOMB0GA1UdDgQWBBTUI4IqOqmoKy5LEjygiJzd2nEGWzAfBgNVHSME
GDAWgBTUI4IqOqmoKy5LEjygiJzd2nEGWzAMBgNVHRMEBTADAQH/MA0GCSqGSIb3
DQEBCwUAA4IBAQC+7TPfDAz1t4yvG4rpypinU2Plp2FOBRdU85CIVREIplbRX8Vv
VMXYySnrKDjVRPcWtY+EtDW91BfS2I2usHkiKcnOYazDoxpamFI3D6lmb82JAAqG
NMV56hIJAYMpVqfbI+vtC93NR3F2fLD8/Hm/X4L8U6kdu2o+vgqYtc4OOl87lhJS
R8xA2N9kihOA1FUMVL89cFgRvxONjAyQImTB8uWl1stUGpH2kE3dABHC1sbZ1dNw
2/OxpfGnj1Qjvi4wKqjGpX/Kqc0AIx2zsKEX9hpPMQK/wlaRg8hlNCXJzzTQ7IVX
7PWJSqn7gpfyZIgZ9JQQ/WieCH32mQ2xGMDD
-----END CERTIFICATE-----
```

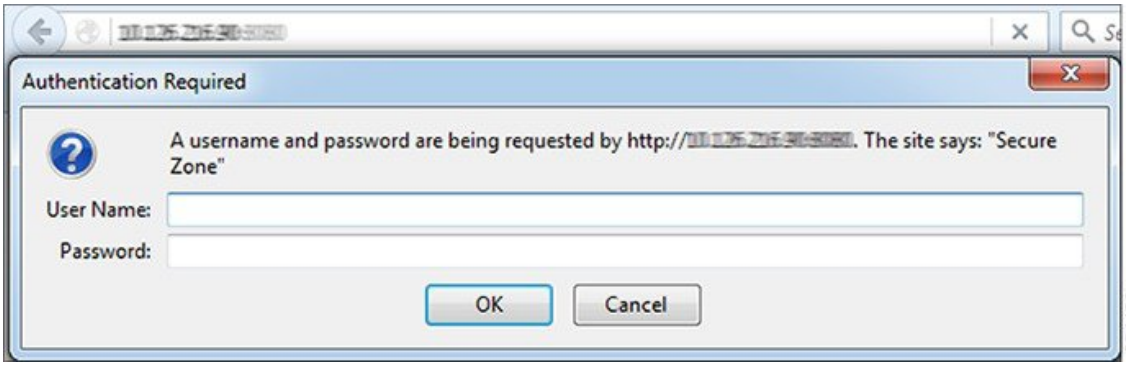
To access the NX-API Developer Sandbox, follow these steps:

Note When using the NX-API Developer Sandbox, we recommend that you use Firefox Release 24.0 or later. The browser must be installed with the latest Adobe Flash player for the **Copy** and **Python** buttons in the NX-API Developer Sandbox to function.

1. Open a browser and enter http://*switch_ip_address*:*port-number* for HTTP, or https://*switch_ip_address*:*port-number* for HTTPS in the address bar.

   The NX-API Developer Sandbox Authentication window is displayed:

   Figure 2: NX-API Developer Sandbox Authentication



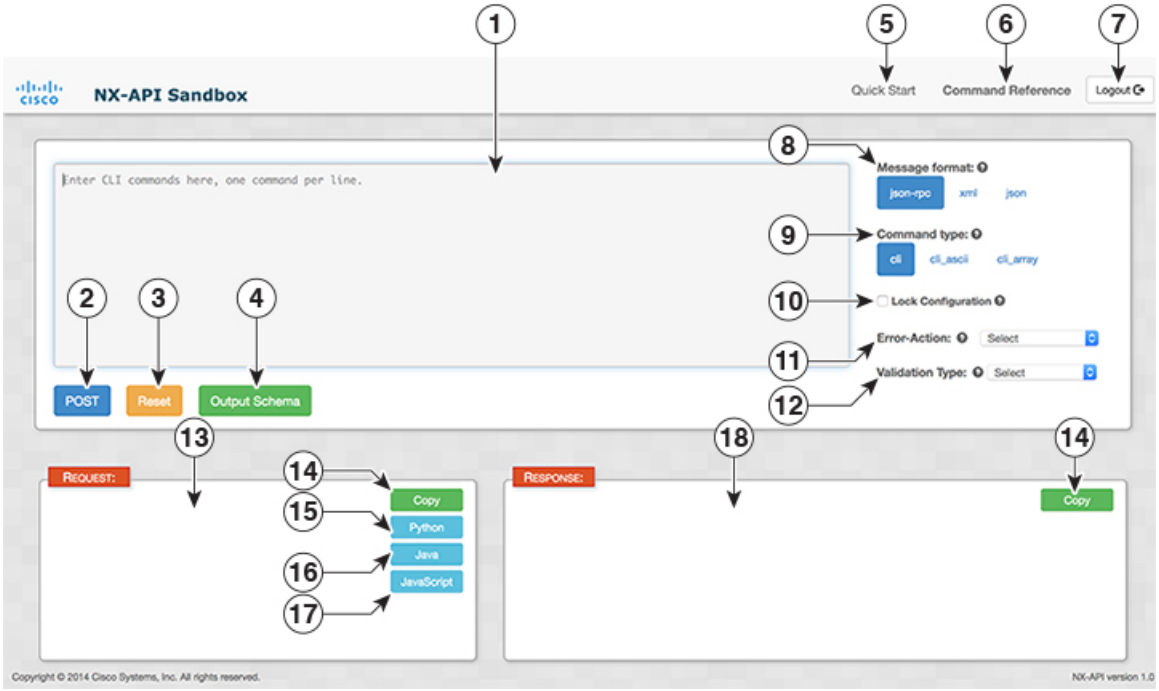
2. Log in using your switch credentials.

   The NX-API Developer Sandbox window is displayed.

The NX-API Developer Sandbox is a web form that is hosted on the switch. It translates NX-OS CLI commands into equivalent XML or JSON payloads, and converts NX-API payloads into their CLI equivalents.

The web form is a single screen with three panes—Command (top pane), Request, and Response—as shown in the Figure 3: NX-API Developer Sandbox.

Figure 3: NX-API Developer Sandbox



1	Command entry—Allows you to enter a command. Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box.	10	Lock Configuration—Provides an exclusive lock to the configuration, whereby no other management agent will be able to modify the configuration.
2	POST—Generates the output for a given command.	11	Error Action—Specifies the error action options • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system was in.
3	Reset—Clears the command and the corresponding output	12	Validation Type—Specifies validation settings. • Validate-Only—Validates the configurations but does not apply the configurations. • Validate-and-Set—Validates the configurations, and applies the configurations on the switch if the validation is successful.
4	Output Schema—Displays the command schema for a command entered in the command pane.	13	REQUEST—Displays the output for a command that is entered in the selected message format.
5	Quick Start—Displays the online help for Cisco MDS NX-API.	14	Copy—Copies the data populated in the **REQUEST** or **RESPONSE** area.

6	Command Reference—Displays the **Command Reference** pane. The **Command Reference** pane displays the command schema details of the command that is selected in the **Show commands** pane. **Note** Supported from Cisco MDS NX-OS Release 8.4(1).	15	Python
7	Logout—Logs the user out of NX-API sandbox.	16	Java **Note** Supported from Cisco MDS NX-OS Release 8.4(1).
8	Message format—Provides different message formats in which the command output is to be displayed.	17	Javascript **Note** Supported from Cisco MDS NX-OS Release 8.4(1).

9	Command type • **cli**— show or configuration commands. • **cli_ascii** — show or configuration commands, output without formatting. • **cli_array** — CLI show commands. CLI show commands that expect structured output. Only for show commands. If the command does not support XML output, an error message is returned. Similar to cli, but with cli_array, data is returned as a list of one element, or an array, within square brackets [ ]. **Note** The **cli_array** command type is supported from Cisco MDS NX-OS Release 8.4(1).	18	RESPONSE—Displays the API response for the command entered in the command entry area.

Controls in the Command pane allow you to choose a message format for a supported API, such as NX-API, and a command type, such as XML or JSON. The available command type options vary depending on the selected message format.

To generate an output of a command using the NX-API Developer Sandbox, follow these steps:

1. Click the Message format type (**json-rpc**, **xml**, **json**) in which the command output is to be displayed. (By default, **json-rpc** is selected.)
2. Click the Command type you have entered. The options differ based on the Message format type selected. (By default, **cli** is selected.)

   You can erase the contents of the text entry box (and the **Request** and **Response** panes) by clicking **Reset** at the bottom of the top pane.

Note

- If you select the **xml** Message format, you can enable chunk mode for the **cli_show** and **cli_show_ascii** Command types. Check the **Enable chunk mode** check box to chunk large **show** command outputs. To view the next chunk of the output, copy the session ID (SID) mentioned in between **<sid>** and **</sid>** tags in the **RESPONSE** area and paste it in the SID box below the **Enable chunk mode** check box.

3. Type or paste NX-OS CLI configuration commands, one command per line, into the text entry box in the top pane.
4. The command that you entered is displayed in the selected Message format in the **REQUEST** area.

   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.

   - To copy the data populated in the **REQUEST** area, click **Copy**.
   - To generate a Python code for the command entered, click **Python**.

   Note The **xml** Message format does not support the **Python** button.

5. Click **POST** to generate the output of the command.

   The output of the command is displayed in the **RESPONSE** area.

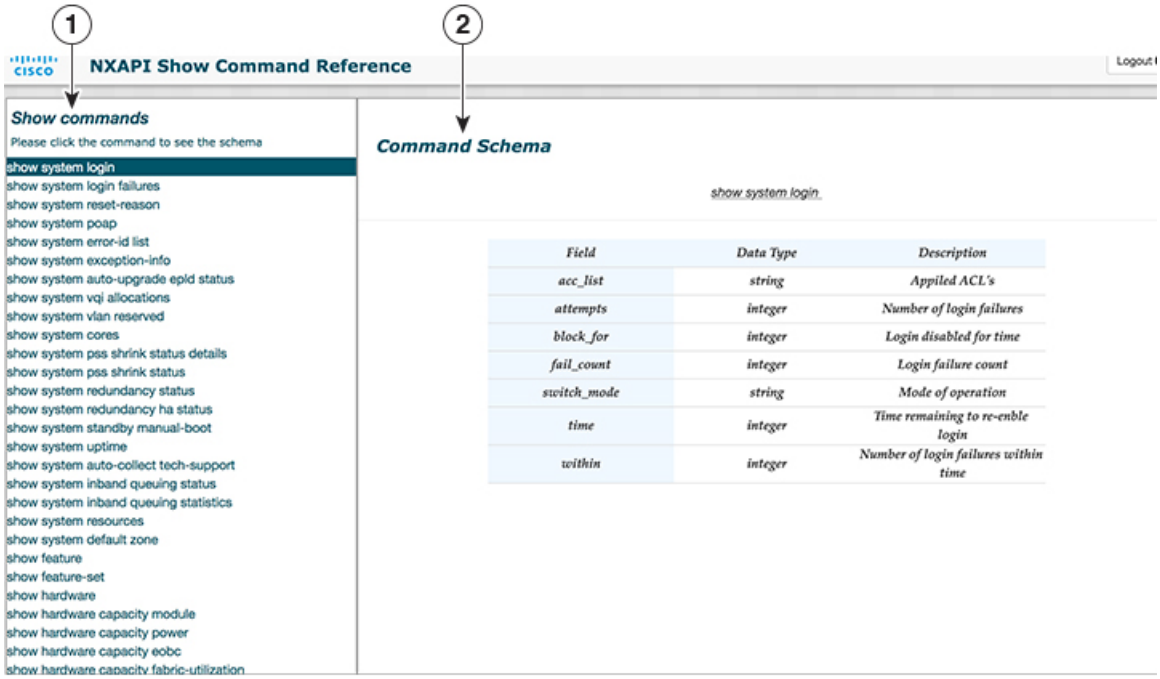
   - To copy the data populated in the **RESPONSE** area, click **Copy**.

   To clear the command and the corresponding output, and reset the page, click **Reset**.
6. Click **Command Reference** to view the list of show commands that are supported for NX-API.

   The **Command Schema** pane displays the details of the command that is selected in the **Show commands** pane.

Note Currently, the **Command Reference** tab supports only the **show** commands.

Figure 4: NX-API Show Command Reference



1	Show commands—Displays the list of supported show commands.
2	Command Schema—Displays the NX-API schema (keywords and description) for a command selected in the **Show commands** pane.

Example: Displaying NX-API Status

The following example displays the NX-API status response in different output formats:

XML Format

`show nxapi`

Request:

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

Response:

```
<ins_api>
  <type>cli_show</type>
```

```
  <version>1.2</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
     <nxapi_status>Enabled</nxapi_status>
     <sandbox_status>Enabled</sandbox_status>
     <http_port>8080</http_port>
    </body>
      <input>show nxapi</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

JSON Format

show nxapi

Request:

```
{
  "ins_api": {
    "version": "1.2",
    "type": "cli_show",
    "chunk": "0",
    "sid": "1",
    "input": "show nxapi",
    "output_format": "json"
  }
}
```

Response:

```
{
  "ins_api": {
    "type": "cli_show",
    "version": "1.2",
    "sid": "eoc",
    "outputs": {
      "output": {
        "input": "show nxapi",
        "msg": "Success",
        "code": "200",
        "body": {
          "nxapi_status": "Enabled",
          "sandbox_status": "Enabled",
          "http_port": "8080"
        }
      }
    }
  }
}
```

JSON-RPC Format

show nxapi

Request:

```
[
  {
    "jsonrpc": "2.0",
```

```
    "method": "cli",
    "params": {
      "cmd": "show nxapi",
      "version": 1.2
    },
    "id": 1
  }
]
```

Response:

```
{
  "jsonrpc": "2.0",
  "result": {
    "body": {
      "nxapi_status": "Enabled",
      "sandbox_status": "Enabled",
      "http_port": "8080"
    }
  },
  "id": 1
}
```

Example: Configuring VSAN to VLAN Mapping

The following example shows how to configure VSAN to VLAN mapping in global configuration mode (**cli_conf**):

```
vlan 3
fcoe vsan 3
vsan database
vsan 3
vsan 3 interface vfc1/8
```

Request:

```
<?xml version="1.0"?>
<ins_api>
  <version>1.2</version>
  <type>cli_conf</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>vlan 3 ;fcoe vsan 3 ;vsan database ;vsan 3 ;vsan 3 interface vfc1/8</input>
  <output_format>xml</output_format>
</ins_api>
```

Response:

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_conf</type>
  <version>1.2</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body/>
      <input>vlan 3</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
    <output>
      <body/>
      <input>fcoe vsan 3</input>
```

```
      <code>200</code>
      <msg>Success</msg>
    </output>
    <output>
      <body/>
      <input>vsan database</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
    <output>
      <body/>
      <input>vsan 3</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
    <output>
      <body/>
      <input>vsan 3 interface vfc1/8</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

Example: Configuring Zones and Zonesets

The following example shows how to configure a zone in global configuration mode (**cli_conf**):

```
zone name zone2 vsan 1
member pwwn 10:00:00:23:45:67:89:ab
member pwwn 10:00:00:23:45:67:89:cd
```

Request:

```
<?xml version="1.0"?>
<ins_api>
  <version>1.2</version>
  <type>cli_conf</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>zone name zone2 vsan 1 ;member pwwn 10:00:00:23:45:67:89:ab ;member pwwn
10:00:00:23:45:67:89:cd</input>
  <output_format>xml</output_format>
</ins_api>
```

Response:

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_conf</type>
  <version>1.2</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body/>
      <input>zone name zone2 vsan 1</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
    <output>
      <body/>
      <input>member pwwn 10:00:00:23:45:67:89:ab</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
```

```
    <output>
      <body/>
      <input>member pwwn 10:00:00:23:45:67:89:cd</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

The following example shows how to configure a zoneset in global configuration mode (**cli_conf**):

```
zoneset name Zoneset1 vsan 1
member zone2
zoneset activate name Zoneset1 vsan 1
```

Request:

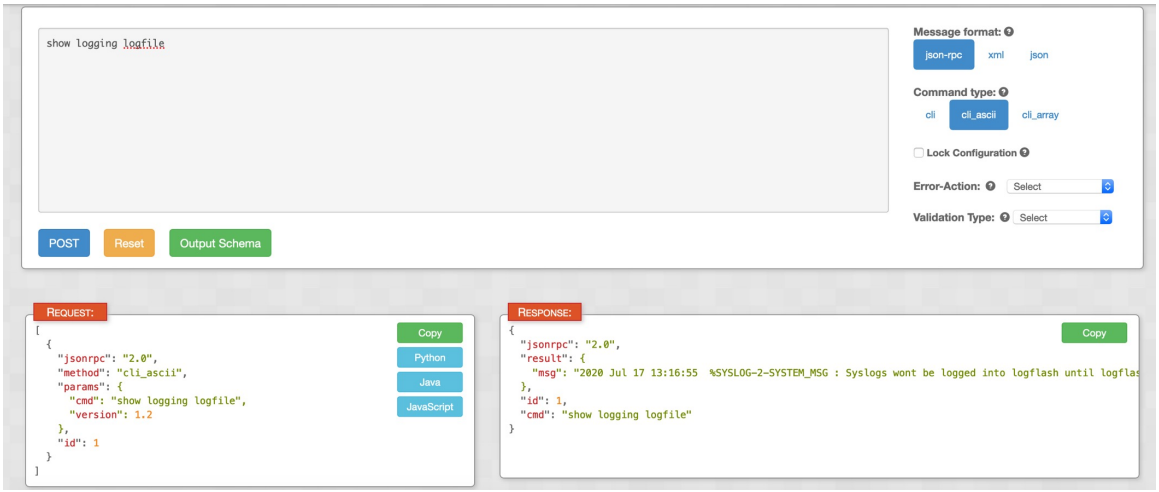
```
<?xml version="1.0"?>
<ins_api>
  <version>1.2</version>
  <type>cli_conf</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>zoneset name Zoneset1 vsan 1 ;member zone2 ;zoneset activate name Zoneset1 vsan
1</input>
  <output_format>xml</output_format>
</ins_api>
```

Response:

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_conf</type>
  <version>1.2</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body/>
      <input>zoneset name Zoneset1 vsan 1</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
    <output>
      <body/>
      <input>member zone2</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
    <output>
      <body>Zoneset activation initiated. check zone status
</body>
      <input>zoneset activate name Zoneset1 vsan 1</input>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

If a **show** command is not NX-API-aware, the output can still be accessed by setting the **Command type** element to **cli_show_ascii** for JSON and XML encoded requests, or **show_ascii** for JSON-RPC encoded requests. The command output is returned in the response body as a single flat string.

The following figure provides an example for a **show** command output that is not NX-API-aware, in the NX-API Developer Sandbox.



NX-API Request Elements

NX-API request elements are sent to a device in XML, JSON, or JSON-RPC formats. The HTTP header of the request must identify the content type of the request.

Note A lock will be released by the system if the session that holds the lock is terminated for any reason. The session that acquired the lock can only perform necessary configurations.

Table 2: NX-API Request Elements for XML or JSON Format

NX-API Request Element	Description
version	Specifies the NX-API version.

NX-API Request Element	Description
type	Specifies the command type to be executed. The following command types are supported: • **cli**—CLI configuration commands. CLI **show** commands that expect structured output. If the command does not support XML output, an error message is returned. • **cli_array**—CLI show commands. CLI **show** commands that expect structured output. Only for show commands. If the command does not support XML output, an error message is returned. Similar to **cli**, but with **cli_array**, data is returned as a list of one element, or an array, within square brackets [ ]. • **cli_ascii** —CLI configuration commands. CLI **show** commands that expect ASCII output. This aligns with existing scripts that parse ASCII output. Users can use existing scripts with minimal changes. • **cli_show** —CLI **show** commands that expect structured output. If the command does not support XML output, an error message is returned. • **cli_show_array**—CLI configuration commands. CLI **show** commands that expect structured output. Only for show commands. Similar to **cli_show**, but with **cli_show_array**, data is returned as a list of one element, or an array, within square brackets [ ]. • **cli_show_ascii** —CLI **show** commands that expect ASCII output. This aligns with existing scripts that parse ASCII output. Users can use existing scripts with minimal changes. • **cli_conf** —CLI configuration commands. **Note** • Each command is executable only with the current user's authority. • A maximum of 10 consecutive **show** commands are supported. If the number of **show** commands exceeds 10, the 11th and subsequent commands are ignored. • No interactive commands are supported.

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. Enable or disable chunk with the following settings: • 0—Do not chunk output. • 1—Chunk output. **Note** • Only **show** commands support chunking. When a series of **show** commands are entered, only the first command is chunked and returned. The output message format is XML, which is the default. Special characters, such as < or >, are converted to form a valid XML message (< is converted to < > is converted to &gt). You can use XML SAX to parse the chunked output. • When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled. **Note** When chunking is enabled, the maximum message size supported is currently 200 MB of the chunked output.
roll_back	Specifies the configuration rollback options. Specify one of the following options. • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in.
validate	Configuration validation settings. This element allows you to validate the commands before you apply them on the switch. This enables you to verify the consistency of a configuration (for example, the availability of necessary hardware resources) before applying it. Choose the validation type from the Validation Type drop-down list. • Validate-Only—Validates the configurations, but does not apply the configurations. • Validate-and-Set —Validates the configurations, and applies the configurations on the switch if the validation is successful.
lock	An exclusive lock on the configuration can be specified, whereby no other management or programming agent will be able to modify the configuration if this lock is held.

NX-API Request Element	Description
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 a *sid* to match the *sid* of the previous response message.
input	Input can be one command or multiple commands. However, commands that belong to different message types should not be mixed. For example, **show** commands belong to the cli_show message format and are not supported in cli_conf message format. **Note** Multiple commands are separated with a semicolon (;). (The ; must be surrounded with single blank characters.) The following are examples of multiple commands: • cli_show **show version ; show interface brief ; show vlan** • cli_conf **interface Eth4/1 ; no shut ; switchport**
output_format	The available output message formats are: • xml—Specifies output in XML format. • json—Specifies output in JSON format. • json-rpc—Specifies output in JSON-RPC format. **Note** The Cisco MDS 9000 device CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch. To manage computational overhead, the JSON output is determined by the amount of output. If the output exceeds 1 MB, the output is returned in XML format. When the output is chunked, only XML output is supported. The content-type header in the HTTP or HTTPS response headers indicate the type of response format (XML, JSON, or JSON-RPC).

NX-API Response Elements

The following table lists the NX-API elements that respond to a CLI command:

Table 3: NX-API Response Elements

NX-API Response Element	Description
version	NX-API version.

NX-API Response Element	Description
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 either of cli_show or cli_show_ascii command type, each command output is enclosed by a single output tag. When the command type is cli_conf, there is a single output tag for all the commands because cli_conf commands require context.
output	Tag that encloses the output of a single command output. For cli_conf command type, this element contains the outputs of all the commands.
input	Tag that encloses a single command specified in the request. This element helps associate a request input element with the appropriate response output element.
body	Body of the command response.
code	Error code returned from command execution. NX-API uses standard HTTP error codes as described by the HTTP Status Code Registry (http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml).
msg	Error message associated with the returned error code.

Table of NX-API Response Codes

The following are the possible NX-API errors, error codes, and messages pertaining to an NX-API response.

Table 4: NX-API Response Codes

NX-API Response	Code	Message
SUCCESS	200	Success.
CUST_OUTPUT_PIPED	204	Output is piped elsewhere due to request.
CHUNK_ALLOW_ONE_CMD_ERR	400	Chunking allowed only to one command.
CLI_CLIENT_ERR	400	CLI execution error.
CLI_CMD_ERR	400	Input CLI command error.
IN_MSG_ERR	400	Request message is invalid.
NO_INPUT_CMD_ERR	400	No input command.

PERM_DENY_ERR	401	Permission denied.
CONF_NOT_ALLOW_SHOW_ERR	405	Configuration mode does not allow **show** command .
SHOW_NOT_ALLOW_CONF_ERR	405	Show mode does not allow configuration.
EXCEED_MAX_SHOW_ERR	413	Maximum number of consecutive **show** commands exceeded. The maximum is 10.
MSG_SIZE_LARGE_ERR	413	Response size too large.
BACKEND_ERR	500	Backend processing error.
FILE_OPER_ERR	500	System internal file operation error.
LIBXML_NS_ERR	500	System internal LIBXML NS error.
LIBXML_PARSE_ERR	500	System internal LIBXML parse error.
LIBXML_PATH_CTX_ERR	500	System internal LIBXML path context error.
MEM_ALLOC_ERR	500	System internal memory allocation error.
USER_NOT_FOUND_ERR	500	User not found from input or cache.
XML_TO_JSON_CONVERT_ERR	500	XML to JSON conversion error.
CHUNK_ALLOW_XML_ONLY_ERR	501	Chunking allows only XML output.
JSON_NOT_SUPPORTED_ERR	501	JSON not supported due to large amount of output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
PIPE_OUTPUT_NOT_SUPPORTED_ERR	501	Pipe operation not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML is not allowed in input.
RESP_BIG_JSON_NOT_ALLOWED_ERR	501	Response has large amount of output. JSON not supported.
STRUCT_NOT_SUPPORTED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Undefined.

Default Settings

The following table lists the default settings for HTTP and HTTPS for Cisco MDS Release versions:

Cisco MDS NX-OS Release	HTTP	HTTPS
Cisco MDS NX-OS Release 8.2(2) and earlier	Enabled	Disabled

Cisco MDS NX-OS Release	HTTP	HTTPS
Cisco MDS NX-OS Release 8.3(1) Cisco MDS NX-OS Release 8.3(2)	Enabled	Enabled
Cisco MDS NX-OS Release 8.4(1) and later	Disabled	Enabled

The Table 5: Supported HTTP and HTTPS Ports, on page 32 table lists the supported HTTP and HTTPS ports for Cisco MDS Release versions:

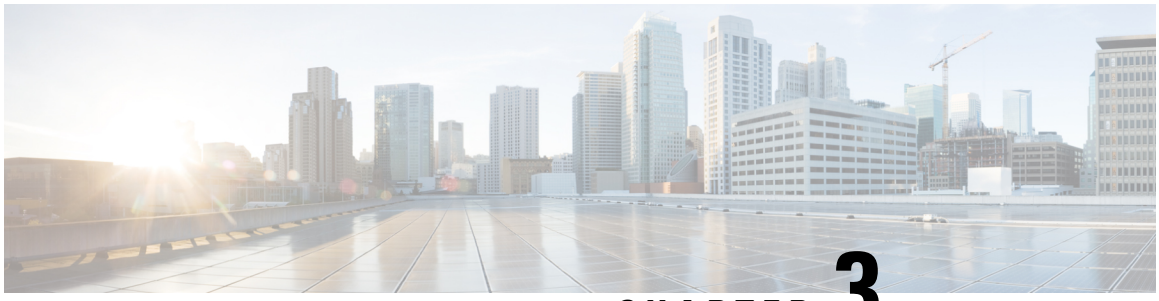
Table 5: Supported HTTP and HTTPS Ports

Cisco MDS NX-OS Release	HTTP Port	HTTPS Port
Cisco MDS NX-OS Release 8.2(1) and earlier	8080	443
Cisco MDS NX-OS Release 8.3(1) and later	8080	8443

Additional References

This section provides additional information related to implementing NX-API.

- NX-API DevNet Community
- MDS NX-API Reference Guide
- NX-API Github (NX-OS Programmability scripts)
- CISCO DCNM API Reference Guide



CHAPTER 3

Python API

- **About the Python API** , on page 33
- Supported Switches, on page 34
- Using Python, on page 34

About the Python API

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

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

www.python.org.

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

The Cisco MDS NX-OS 9000 Series devices support Python v2.7.5 in both interactive and non-interactive (script) modes.

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

The Python scripting capability is also used in the Overlay CLI feature of SAN Analytics to interpret the analytics data.

The Python interpreter is available in the Cisco MDS software.

Note From Cisco MDS NX-OS Release 8.2(1), Python version 2.7.8 is supported.

Starting with Cisco MDS NX-OS Release 8.4(2), Python version 3.7.3 is supported. Python2.7 is end of support, and future Cisco MDS NX-OS software will deprecate the Python2.7 support. We recommend using 'python3' for new scripts.

Supported Switches

Python is supported on the following Cisco MDS 9000 Series Switches:

- Cisco MDS 9700 Series Switches

  All Fibre Channel Switching Modules

- Cisco MDS 9132T Multilayer Fabric Switch
- Cisco MDS 9148T Multilayer Fabric Switch
- Cisco MDS 9396T Multilayer Fabric Switch

Note Python API is not supported on Cisco MDS 16 Gbps Fabric switches such as Cisco MDS 9148S, Cisco MDS 9250i, and Cisco MDS 9396S.

Using Python

This section describes how to write and execute Python scripts.

Cisco Python Package

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

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

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

NAME
    cisco

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

PACKAGE CONTENTS
    acl
    bgp
    cisco_secret
    cisco_socket
    feature
    interface
    key
    line_parser
    md5sum
    nxcli
```

```
    ospf
    routemap
    routes
    section_parser
    ssh
    system
    tacacs
    vrf

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

Using the CLI Command APIs

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

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

Table 6: CLI Command APIs

API	Description
cli() Example: `string = cli ("cli-command")`	Returns the raw output of CLI commands, including control/special characters. **Note** The interactive Python interpreter prints control/special characters 'escaped'. A carriage return is printed as '\n' and gives results that might be difficult to read. The **clip()** API gives results that are more readable.
clid() Example: `json_string = clid ("cli-command")`	For CLI commands that support XML, this API returns JSON output. **Note** An exception is thrown when XML is not used. This API can be useful when searching the output of show commands.

clip()
Example:
`clip ("cli-command")`
	Prints the output of the CLI command directly to stdout and returns nothing to Python.
	Note `clip ("cli-command")`
	is equivalent to
	`r=cli("cli-command")`
	`print r`

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

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

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

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

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

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

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

Invoking the Python Interpreter from the CLI

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

Note

- The Python interpreter is designated with the ">>>" or "…" prompt.
- In Cisco MDS NX-OS Release 7.3(x) and later releases, the Python interpreter can be invoked only in Privileged EXEC mode on Cisco MDS 9700 Series Switches.

The following example shows how to invoke Python2 .7.5 from the CLI:

```
switch# python

Python 2.7.5 (default, Jun  3 2016, 03:57:06)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import *
>>> cli("show clock")
'Time source is NTP\n06:32:20.023 UTC Tue Jan 31 2017\n'
>>> exit()
```

The following example shows how to invoke Python3 from the CLI:

```
switch# python3

Python 3.7.3 (default, Aug 26 2019, 23:20:10)
[GCC 4.6.3] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import *
>>> cli("show clock")
'Time source is NTP\n07:46:50.923 UTC Tue Apr 28 2020\n'
```

Display Formats

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

Example 1:

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

Example 2:

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

Example 3:

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

Example 4:

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
...   print "%30s = %s" % (k, out[k])
...
               kern_uptm_secs = 6
               kick_file_name = bootflash:///n9000-dk9.6.1.2.I1.1.bin
                   rr_service = None
                    module_id = Supervisor Module
                  kick_tmstmp = 10/21/2013 00:06:10
               bios_cmpl_time = 08/17/2013
               bootflash_size = 20971520
            kickstart_ver_str = 6.1(2)I1(2) [build 6.1(2)I1(2)] [gdb]
               kick_cmpl_time = 10/20/2013 4:00:00
                   chassis_id = Nexus9000 C9508 (8 Slot) Chassis
                proc_board_id = SAL171211LX
                       memory = 16077872
                 manufacturer = Cisco Systems, Inc.
               kern_uptm_mins = 26
                 bios_ver_str = 06.14
                     cpu_name = Intel(R) Xeon(R) CPU E5-2403
                kern_uptm_hrs = 2
```

```
                      rr_usecs = 816550
                    rr_sys_ver = None
                     rr_reason = Reset Requested by CLI command reload
                      rr_ctime = Mon Oct 21 00:10:24 2013
                    header_str = Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Documents: http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home.html
Copyright (c) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained herein are owned by
other third parties and are used and distributed under license.
Some parts of this software are covered under the GNU Public
License. A copy of the license is available at
http://www.gnu.org/licenses/gpl.html.
                      host_name = switch
                       mem_type = kB
                kern_uptm_days = 0
>>>
```

Non-interactive Python

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

The Cisco MDS 9000 Series device also supports the source CLI command for running Python scripts. The `bootflash:scripts` directory is the default script directory for the source CLI command.

Note To run Python scripts using the Python3 interpreter, ensure the first line of the script contains python3 string such as, #!/isan/bin/python3 OR #!/usr/bin/env python3.

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

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

from cli import *
import sys, time

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

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

i = 0
while (i < count):
  time.sleep(delay)
```

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

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

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

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

policy-map type network-qos nw-pfc
policy-map type network-qos no-drop-2
policy-map type network-qos wred-policy
policy-map type network-qos pause-policy
policy-map type qos foo
policy-map type qos classify
policy-map type qos cos-based
policy-map type qos no-drop-2
policy-map type qos pfc-tor-port
```

Running Scripts with Embedded Event Manager

On Cisco MDS 9000 Series devices, embedded event manager (EEM) policies support Python scripts.

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

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

  ```
  switch# show running-config eem

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

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

- You can search for the action triggered by the event in the log file by running the **show file** *logflash*:*event_archive_1* command.

```
switch# show file logflash:event_archive_1 | last 33

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

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

PC_VSH_CMD_TLV(7679) with q
```

Python Integration with Cisco MDS NX-OS Network Interfaces

On Cisco MDS 9000 Series devices, Python is integrated with the underlying Cisco MDS NX-OS network interfaces.

The following example shows how to retrieve an HTML document over the management interface of a Cisco MDS 9000 Series device. You can also establish a connection to an external entity over the inband interface.

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import urllib2
>>> from cisco.vrf import *
>>> page=urllib2.urlopen('http://172.23.40.211:8000/welcome.html')
>>> print page.read()
Hello Cisco Nexus 9000

>>>
```

Cisco MDS NX-OS Security with Python

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

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

Examples of Security and User Authority

The following example shows how a privileged user runs commands:

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

```
>>> f.close()
>>> r=open('/tmp/test','r')
>>> print r.read()
hello from python
>>> r.close()
```

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

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
system(whoami): rejected!
-1
>>> f=open('/tmp/test','r')
Permission denied. Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 13] Permission denied: '/tmp/test'
>>>
```

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

The following is an example for a privileged user:

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

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

version 6.1(2)I2(1)

interface Ethernet1/48
  vrf member blue

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

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

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

The following example shows an RBAC configuration:

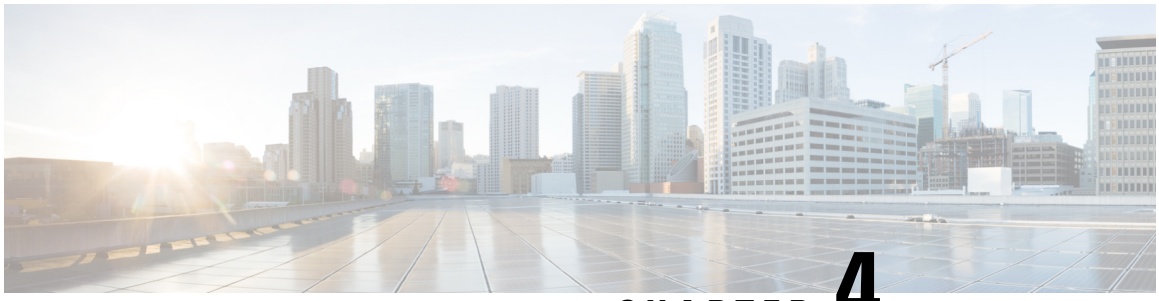
```
switch# show user-account
user:admin
        this user account has no expiry date
        roles:network-admin
user:pyuser
        this user account has no expiry date
        roles:network-operator python-role
switch# show role name python-role
```

Example of Running Script with Scheduler

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

```
#!/bin/env python
from cli import *
from nxos import *
import os

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

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

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



CHAPTER 4

Ansible

Ansible is an open-source IT automation engine that automates cloud provisioning, configuration management, application deployment, intraservice orchestration, and other IT needs. Similar to Puppet, and Chef, Ansible enables administrators to manage, automate, and orchestrate various types of server environments. Ansible is agentless, and does not require a software agent to be installed on the target node (server or switch) in order to automate the device. By default, Ansible requires SSH and Python support on the target node, but Ansible can also be easily extended to use any API. Ansible playbooks are written in YAML, that allows you to describe your automation jobs in an easily readable format. Inside each Ansible playbook, we can use various Ansible modules which make API calls towards Cisco MDS switches.

Ansible modules make API calls against the NX-API to gather real-time state data and to make configuration changes on the Cisco Nexus devices. For more information about Ansible, see Ansible's official documentation.

Note

- For Cisco MDS Anisble modules, you do not need a Python interpreter on the target node.
- Enable SSH on the device using the **feature ssh** command.

For more information on the Cisco MDS modules supported on Ansible, see the Ansible Modules.

- Getting Started, on page 43
- Host File, on page 43
- Documentation, on page 44
- Example Playbook, on page 44

Getting Started

For information on Ansible installation, refer to the official Anisble installation guide https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html.

Host File

The host file is where the devices under management are listed. A single device can be in a single group or included in multiple groups. In the below host file, a single group called leaf, which has 2 devices, mds1 and mds2 are used. The connection is set to local as it is connected through CLI to manage the devices, and the

username and password is stored in the host file. For further security, this host file can be encrypted using the Ansible Vault, which we are not utilizing here.

```
$ cat /etc/ansible/hosts
[all:vars]
ansible_connection = local
un=username
pwd=password

[leaf]
mds1
mds2
```

Documentation

Documentation for all Cisco MDS NX-OS modules can be found at https://docs.ansible.com/ansible/latest/collections/cisco/nxos/ or alternatively from the terminal, by utilizing the inbuilt documentation tool.

```
$ansible-doc
```

Example Playbook

In this initial playbook, we will provision a couple of VSANs and also delete a VSAN. We will use the Ansible module called **nxos_vsan** to automate this task.

As you can see below, the playbook is defined in YAML.

```
---
- name: VSAN TEST (NXOS)
  gather_facts: no
  hosts:
     - mds1
  vars:
    creds:
      host: "{{ inventory_hostname }}"
      username: "{{ un }}"
      password: "{{ pwd }}"
      transport: cli

  tasks:
    - name: Test that vsan module works
      nxos_vsan:
        provider: "{{ creds }}"
        vsan:
           - id: 922
             name: vsan-SAN-A
             suspend: False
             interface:
                - fc1/1
                - fc1/2
                - port-channel 1
             remove: False
           - id: 923
             name: vsan-SAN-B
             suspend: True
             interface:
                - fc1/11
                - fc1/21
                - port-channel 2
```

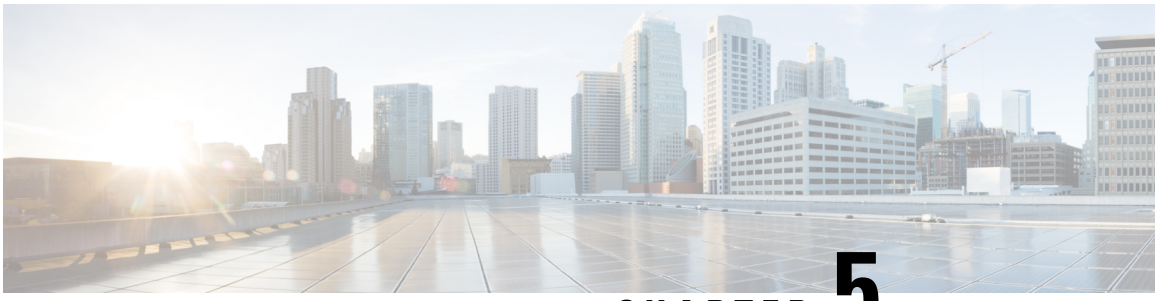
```
            remove: False
          - id: 1923
            name: vsan-SAN-Old
            remove: True
      register: result
    - debug: var=result
```

Assuming the above playbook is called **vsan.yml**, this task can then be run from the terminal as shown below.

```
$ ansible-playbook vsan.yml

PLAY [VSAN TEST (NXOS)] ********************************************************

TASK [Test that vsan module works] *********************************************
changed: [mds1]

TASK [debug] *******************************************************************
ok: [mds1] => {
    "result": {
        "changed": true,
        "cmds": [
            "terminal dont-ask",
            "vsan database",
            "vsan 922",
            "vsan 922 name vsan-SAN-A",
            "no vsan 922 suspend",
            "vsan database",
            "vsan 922 interface fc1/1",
            "vsan 922 interface fc1/2",
            "vsan 922 interface port-channel 1",
            "vsan database",
            "vsan 923",
            "vsan 923 name vsan-SAN-B",
            "vsan 923 suspend",
            "vsan database",
            "vsan 923 interface fc1/11",
            "vsan 923 interface fc1/21",
            "vsan 923 interface port-channel 2",
            "vsan database",
            "no vsan 1923",
            "no terminal dont-ask"
        ],
        "failed": false,
        "messages": [
            "creating vsan 922",
            "setting vsan name to vsan-SAN-A for vsan 922",
            "no suspending the vsan 922",
            "adding interface fc1/1 to vsan 922",
            "adding interface fc1/2 to vsan 922",
            "adding interface port-channel 1 to vsan 922",
            "creating vsan 923",
            "setting vsan name to vsan-SAN-B for vsan 923",
            "suspending the vsan 923",
            "adding interface fc1/11 to vsan 923",
            "adding interface fc1/21 to vsan 923",
            "adding interface port-channel 2 to vsan 923",
            "deleting the vsan 1923"
        ]
    }
}

PLAY RECAP *********************************************************************
mds1                       : ok=2    changed=1    unreachable=0    failed=0
```

Conclusion

We have just seen how we can configure/unconfigure VSANs using Ansible, this is a simple example but the modules can be used in a variety of tasks that needs automation.



CHAPTER 5

Cisco MDS SDK

Cisco MDS-SDK is a Python-based library for the Cisco MDS Switches.

This library is useful for automating day-to-day tasks or developing new tools which involve Cisco MDS Switches.

Cisco MDS-SDK leverages NX-API to communciate with devices, but it can also use SSH for backward compatibility. We recommend that you use NX-API for switches running Cisco MDS NX-OS Release 8.4(2a) or later.

For more information on the Cisco MDS-SDK, see the GitHub page at, https://github.com/Cisco-SAN/mdssdk.

For Cisco MDS-SDK documentation, see http://mdssdk.readthedocs.io.