NX-API CLI

About NX-API CLI

NX-API CLI is an enhancement to the Cisco NX-OS CLI system, which supports XML output. NX-API CLI also supports JSON output format for specific commands.

On Cisco Nexus switches, command-line interfaces (CLIs) are run only on the switch. NX-API CLI improves the accessibility of these CLIs by making them available outside of the switch by using HTTP/HTTPS. You can use this extension to the existing Cisco NX-OS CLI system on the switches. NX-API CLI supports **show** commands, configurations, and Linux Bash.

NX-API CLI supports JSON-RPC.

Transport

NX-API uses HTTP/HTTPS as its transport. CLIs are encoded into the HTTP/HTTPS POST body.

NX-API is also supported through UNIX Domain Sockets for applications running natively on the host or within Guest Shell.

The NX-API backend uses the Nginx HTTP server. The Nginx process, and all its children processes, are under the Linux cgroup protection where the CPU and memory usage is capped. If the Nginx memory usage exceeds the cgroup limitations, the Nginx process is restarted and the NX-API configuration (the VRF, port, and certificate configurations) is restored.

Message Format

NX-API is an enhancement to the Cisco Nexus 7000 Series CLI system, which supports XML output. NX-API also supports JSON output format for specific commands.

NX-API is an enhancement to the Cisco NX-OS CLI system, which supports XML output. NX-API also supports JSON output format for specific commands.

Note

- NX-API XML output presents information in a user-friendly format.
- NX-API XML does not map directly to the Cisco NX-OS NETCONF implementation.
- NX-API XML output can be converted into JSON.

Security

- NX-API supports HTTPS. All communication to the device is encrypted when you use HTTPS.

NX-API is integrated into the authentication system on the device. Users must have appropriate accounts to access the device through NX-API. NX-API uses HTTP basic authentication. All requests must contain the username and password in the HTTP header.

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

You can enable NX-API by using the **feature** manager CLI command. NX-API is disabled by default.

NX-API provides a session-based cookie, **nxapi_auth** when users first successfully authenticate. With the session cookie, the username and password are included in all subsequent NX-API requests that are sent to the device. The username and password are used with the session cookie to bypass performing the full authentication process again. If the session cookie is not included with subsequent requests, another session cookie is required and is provided by the authentication process. Avoiding unnecessary use of the authentication process helps to reduce the workload on the device.

Note A **nxapi_auth** cookie expires in 600 seconds (10 minutes). This value is a fixed and cannot be adjusted.

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

Using NX-API CLI

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

Note For more details about NX-API response codes, see Table of NX-API Response Codes, on page 17.

NX-API CLI is enabled by default for local access. The remote HTTP access is disabled by default.

The following example shows how to configure and launch the NX-API CLI:

- Enable the management interface.

```
switch# conf t
Enter configuration commands, one per line.
End with CNTL/Z.
switch(config)# interface mgmt 0
switch(config-if)# ip address 10.126.67.53/25
switch(config-if)# vrf context managment
switch(config-vrf)# ip route 0.0.0.0/0 10.126.67.1
switch(config-vrf)# end
switch#
```

- Enable the NX-API **nxapi** feature.

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

The following example shows a request and its response in XML format:

Request:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ins_api>
  <version>0.1</version>
  <type>cli_show</type>
  <chunk>0</chunk>
  <sid>session1</sid>
  <input>show switchname</input>
  <output_format>xml</output_format>
</ins_api>
```

Response:

```
<?xml version="1.0"?>
<ins_api>
  <type>cli_show</type>
  <version>0.1</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>
        <hostname>switch</hostname>
      </body>
      <input>show switchname</input>
      <msg>Success</msg>
      <code>200</code>
    </output>
  </outputs>
</ins_api>
```

The following example shows a request and its response in JSON format:

Request:

```
{
    "ins_api": {
        "version": "0.1",
        "type": "cli_show",
        "chunk": "0",
        "sid": "session1",
        "input": "show switchname",
        "output_format": "json"
```

```
    }
}
```

Response:

```
{
    "ins_api": {
        "type": "cli_show",
        "version": "0.1",
        "sid": "eoc",
        "outputs": {
            "output": {
                "body": {
                    "hostname": "switch"
                },
                "input": "show switchname",
                "msg": "Success",
                "code": "200"
            }
        }
    }
}
```

Note There is a known issue where an attempt to delete a user might fail, resulting in an error message similar to the following appearing every 12 hours or so:

`user delete failed for` *`username`*`:userdel: user` *`username`* `is currently logged in - securityd`

This issue might occur in a scenario where you try to delete a user who is still logged into a switch through NX-API. Enter the following command in this case to try to log the user out first:

```
switch(config)# clear user username
```

Then try to delete the user again. If the issue persists after attempting this workaround, contact Cisco TAC for further assistance.

Escalate Privileges to Root on NX-API

For NX-API, the privileges of an admin user can escalate their privileges for root access.

The following are guidelines for escalating privileges:

- Only an admin user can escalate privileges to root.
- Escalation to root is password protected.

The following examples show how an admin escalates privileges to root and how to verify the escalation. Note that after becoming root, the **whoami** command shows you as admin; however, the admin account has all the root privileges.

First example:

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
```

```
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo su root ; whoami</input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>admin </body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

Second example:

```
<?xml version="1.0"?>
<ins_api>
  <version>1.0</version>
  <type>bash</type>
  <chunk>0</chunk>
  <sid>sid</sid>
  <input>sudo cat path_to_file </input>
  <output_format>xml</output_format>
</ins_api>

<?xml version="1.0" encoding="UTF-8"?>
<ins_api>
  <type>bash</type>
  <version>1.0</version>
  <sid>eoc</sid>
  <outputs>
    <output>
      <body>[Contents of file]</body>
      <code>200</code>
      <msg>Success</msg>
    </output>
  </outputs>
</ins_api>
```

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: Cisco Nexus 9000 NX-OS NX-API

NX-API Management Commands

You can enable and manage NX-API with the CLI commands listed in the following table.

Table 1: NX-API Management Commands

NX-API Management Command	Description
feature nxapi	Enables NX-API.
no feature nxapi	Disables NX-API.
nxapi {**http** | **https**} **port** *port*	Specifies a port.
no nxapi {**http** | **https**}	Disables HTTP/HTTPS.
show nxapi	Displays port and certificate information.

nxapi certificate {httpscrt certfile | **httpskey keyfile**} *filename*	Specifies the upload of the following:
	• HTTPS certificate when httpscrt is specified.
	• HTTPS key when httpskey is specified.
	Example of HTTPS certificate:
	`nxapi certificate httpscrt certfile bootflash:cert.crt`
	Example of HTTPS key:
	`nxapi certificate httpskey keyfile bootflash:privkey.key`

nxapi certificate enable	Enables a certificate.

NX-API Management Command	Description
nxapi certificate sudi	This CLI provides a secure way of authenticating to the device by using Secure Unique Device Identifier (SUDI). The SUDI based authentication in nginx will be used by the CISCO SUDI compliant controllers. SUDI is an IEEE 802.1AR-compliant secure device identity in an X.509v3 certificate which maintains the product identifier and serial number of Cisco devices. The identity is implemented at manufacturing and is chained to a publicly identifiable root certificate authority. **Note** • When NX-API comes up with the SUDI certificate, it is not accessible by any third-party applications like browser, curl, and so on. • "nxapi certificate sudi" will overwrite the custom certificate/key if configured, and there is no way to get the custom certificate/key back. • "nxapi certificate sudi" and "nxapi certificate trustpoint" and "nxapi certificate enable" are mutually exclusive , and configuring one will delete the other configuration. • NX-API do not support SUDI certificate-based client certificate authentication. If client certificate authentication is needed, then Identity certificate need to be used. • As NX-API certificate CLI is not present in show run output, CR/Rollback case currently does not go back to the custom certificate once it is overwritten with "nxapi certificate sudi" options.
nxapi use-vrf *vrf*	Specifies the default VRF, management VRF, or named VRF. **Note** In Cisco NX-OS Release 7.0(3)I2(1) NGINX listens on only one VRF.

NX-API Management Command	Description
ip netns exec management iptables	Implements any access restrictions and can be run in management VRF. **Note** You must enable **feature bash-shell** and then run the command from Bash Shell. For more information on Bash Shell, see the chapter on Bash. *Iptables is a command-line firewall utility that uses policy chains to allow or block traffic and almost always comes pre-installed on any Linux distribution.* **Note** For more information about making iptables persistent across reloads when they are modified in a bash-shell, see Making an Iptable Persistent Across Reloads, on page 16.

Following is an example of a successful upload of an HTTPS certificate:

```
switch(config)# nxapi certificate httpscrt certfile certificate.crt
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

Note You must configure the certificate and key before enabling the certificate.

Following is an example of a successful upload of an HTTPS key:

```
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
Upload done. Please enable. Note cert and key must match.
switch(config)# nxapi certificate enable
switch(config)#
```

Working With Interactive Commands Using NX-API

To disable confirmation prompts on interactive commands and avoid timing out with an error code 500, prepend interactive commands with **terminal dont-ask**. Use **;** to separate multiple interactive commands, where each **;** is surrounded with single blank characters.

Following are several examples of interactive commands where **terminal dont-ask** is used to avoid timing out with an error code 500:

```
terminal dont-ask ; reload module 21
```

```
terminal dont-ask ; system mode maintenance
```

NX-API Request Elements

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

You use the NX-API elements that are listed in the following table to specify a CLI command:

Note Users need to have permission to execute "configure terminal" command. When JSON-RPC is the input request format, the "configure terminal" command will always be executed before any commands in the payload are executed.

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 type of command to be executed. The following types of commands are supported: • **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 **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 are able to use existing scripts with minimal changes. • **cli_conf** CLI configuration commands. • **bash** Bash commands. Most non-interactive Bash commands are supported by NX-API. **Note** • Each command is only executable with the current user's authority. • The pipe operation is supported in the output when the message type is ASCII. If the output is in XML format, the pipe operation is not supported. • 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.

<table>
<tr><th>NX-API Request Element</th><th>Description</th></tr>
<tr><td>chunk</td><td>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:

Note
<table><tr><td>0</td><td>Do not chunk output.</td></tr><tr><td>1</td><td>Chunk output.</td></tr></table>
Note
• Only show commands support chunking. When a series of show commands are entered, only the first command is chunked and returned.
• For the XML output message format (XML is the default.), special characters, such as < or >, are converted to form a valid XML message (< is converted into < > is converted into &gt).

You can use XML SAX to parse the chunked output.

Note
When chunking is enabled, the message format is limited to XML. JSON output format is not supported when chunking is enabled.</td></tr>
<tr><td>rollback</td><td>Valid only for configuration CLIs, not for show commands. 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.

Note
The rollback element is available in the cli_conf mode when the input request format is XML or JSON.</td></tr>
<tr><td>sid</td><td>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.</td></tr>
</table>

<table>
<tr><th>NX-API Request Element</th><th>Description</th></tr>
<tr><td>input</td><td>Input can be one command or multiple commands. However, commands that belong to different message types should not be mixed. For example, show commands are cli_show message type and are not supported in cli_conf mode.
Note
Except for bash, multiple commands are separated with " ; ". (The ; must be surrounded with single blank characters.)
Prepend commands with <code>terminal dont-ask</code> to avoid timing out with an error code 500. For example:
<code>terminal dont-ask ; cli_conf ; interface Eth4/1 ; no shut ; switchport</code>
For bash, multiple commands are separated with ";". (The ; is not surrounded with single blank characters.)
The following are examples of multiple commands:
Note
<table>
<tr><td>cli_show</td><td><code>show version ; show interface brief ; show vlan</code></td></tr>
<tr><td>cli_conf</td><td><code>interface Eth4/1 ; no shut ; switchport</code></td></tr>
<tr><td>bash</td><td><code>cd /bootflash;mkdir new_dir</code></td></tr>
</table></td></tr>
<tr><td>output_format</td><td>The available output message formats are the following:
Note
<table>
<tr><td>xml</td><td>Specifies output in XML format.</td></tr>
<tr><td>json</td><td>Specifies output in JSON format.</td></tr>
</table>
Note
The Cisco NX-OS CLI supports XML output, which means that the JSON output is converted from XML. The conversion is processed on the switch.
To manage the 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/HTTPS headers indicate the type of response format (XML or JSON).</td></tr>
</table>

When JSON-RPC is the input request format, use the NX-API elements that are listed in the following table to specify a CLI command:

Table 3: NX-API Request Elements for JSON-RPC Format

NX-API Request Element	Description
jsonrpc	A string specifying the version of the JSON-RPC protocol. Version must be 2.0.
method	A string containing the name of the method to be invoked. NX-API supports either: • **cli**–show or configuration commands • **cli_ascii**–show or configuration commands; output without formatting • **cli_array**–only for show commands; similar to **cli**, but with **cli_array**, data is returned as a list of one element, or an array, within square brackets, [ ].
params	A structured value that holds the parameter values used during the invocation of a method. It must contain the following: • **cmd**–CLI command • **version**–NX-API request version identifier
rollback	Valid only for configuration CLIs, not for show commands. Configuration rollback options. You can specify one of the following options. • Stop-on-error—Stops at the first CLI that fails. • Continue-on-error—Ignores the failed CLI and continues with other CLIs. • Rollback-on-error—Performs a rollback to the previous state the system configuration was in.
id	An optional identifier established by the client that must contain a string, number, or null value, if it is specified. The value should not be null and numbers contain no fractional parts. If a user does not specify the id parameter, the server assumes that the request is simply a notification, resulting in a no response, for example, *id* : 1

NX-API Response Elements

The NX-API elements that respond to a CLI command are listed in the following table:

Table 4: NX-API Response Elements

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. This element helps associate a request input element with the appropriate response output element.
body	Body of the command response.
code	Error code returned from the command execution. NX-API uses standard HTTP error codes as described by the Hypertext Transfer Protocol (HTTP) Status Code Registry (http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml).
msg	Error message associated with the returned error code.

Restricting Access to NX-API

There are two methods for restricting HTTP and HTTPS access to a device: ACLs and iptables. The method that you use depends on whether you have configured a VRF for NX-API communication using the `nxapi use-vrf <vrf-name>` CLI command.

Use ACLs to restrict HTTP or HTTPS access to a device only if you have not configured NXAPI to use a specific VRF. For information about configuring ACLs, see the *Cisco Nexus Series NX-OS Security Configuration Guide* for your switch family.

If you have configured a VRF for NX-API communication, however, ACLs will not restrict HTTP or HTTPS access. Instead, create a rule for an iptable. For more information about creating a rule, see Updating an iptable, on page 15.

Updating an iptable

An iptable enables you to restrict HTTP or HTTPS access to a device when a VRF has been configured for NX-API communication. This section demonstrates how to add, verify, and remove rules for blocking HTTP and HTTPS access to an existing iptable.

Procedure

Step 1 To create a rule that blocks HTTP access:

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp  --dport 80 -j DROP
```

Note
The **management** mentioned in this step is the VRF name. It can be **management | default | custom vrf name**.

Step 2 To create a rule that blocks HTTPS access:

```
bash-4.3# ip netns exec management iptables -A INPUT -p tcp  --dport 443 -j DROP
```

Step 3 To verify the applied rules:

```
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere             tcp dpt:http
DROP       tcp  --  anywhere             anywhere             tcp dpt:https

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Step 4 To create and verify a rule that blocks all traffic with a 10.155.0.0/24 subnet to port 80:

```
bash-4.3# ip netns exec management iptables -A INPUT -s 10.155.0.0/24 -p tcp --dport 80 -j
 DROP
bash-4.3# ip netns exec management iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  10.155.0.0/24        anywhere             tcp dpt:http

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Step 5 To remove and verify previously applied rules:

This example removes the first rule from INPUT.

```
bash-4.3# ip netns exec management iptables -D INPUT 1
bash-4.3# ip netns exec management iptables -L
```

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

What to do next

The rules in iptables are not persistent across reloads when they are modified in a bash-shell. To make the rules persistent, see Making an Iptable Persistent Across Reloads, on page 16.

Making an Iptable Persistent Across Reloads

The rules in iptables are not persistent across reloads when they are modified in a bash-shell. This section explains how to make a modified iptable persistent across a reload.

Before you begin

You have modified an iptable.

Procedure

Step 1 Create a file called iptables_init.log in the /etc directory with full permissions:

```
bash-4.3# touch /etc/iptables_init.log; chmod 777 /etc/iptables_init.log
```

Step 2 Create the /etc/sys/iptables file where your iptables changes will be saved:

```
bash-4.3# ip netns exec management iptables-save > /etc/sysconfig/iptables
```

Step 3 Create a startup script called iptables_init in the /etc/init.d directory with the following set of commands:

```
#!/bin/sh

### BEGIN INIT INFO

# Provides:          iptables_init

# Required-Start:

# Required-Stop:

# Default-Start:     2 3 4 5

# Default-Stop:

# Short-Description: init for iptables

# Description:       sets config for iptables

#                    during boot time
```

```
### END INIT INFO


PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
start_script() {
    ip netns exec management iptables-restore < /etc/sysconfig/iptables
    ip netns exec management iptables
    echo "iptables init script executed" > /etc/iptables_init.log
}
case "$1" in
  start)
    start_script
   ;;
  stop)
    ;;
  restart)
    sleep 1
    $0 start
    ;;
  *)
    echo "Usage: $0 {start|stop|status|restart}"
    exit 1
esac
exit 0
```

Step 4 Set the appropriate permissions to the startup script:

```
bash-4.3# chmod 777 /etc/init.d/iptables_int
```

Step 5 Set the iptables_int startup script to on with the chkconfig utility:

```
bash-4.3# chkconfig iptables_init on
```

The iptables_init startup script will now execute each time that you perform a reload, making the iptable rules persistent.

Table of NX-API Response Codes

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

Note The standard HTTP error codes are at the Hypertext Transfer Protocol (HTTP) Status Code Registry (http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml).

Table 5: NX-API Response Codes

NX-API Response	Code	Message
SUCCESS	200	Success.
CUST_OUTPUT_PIPED	204	Output is piped elsewhere due to request.
BASH_CMD_ERR	400	Bash command error.

CHUNK_ALLOW_ONE_CMD_ERR	400	Chunking honors only one command.
CLI_CLIENT_ERR	400	CLI execution error.
CLI_CMD_ERR	400	Input CLI command error.
IN_MSG_ERR	400	Incoming 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** .
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. This is a request format error.
LIBXML_PARSE_ERR	500	System internal LIBXML parse error. This is a request format error.
LIBXML_PATH_CTX_ERR	500	System internal LIBXML path context error. This is a request format 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.
BASH_CMD_NOT_SUPPORTED_ERR	501	Bash command not supported.
CHUNK_ALLOW_XML_ONLY_ERR	501	Chunking allows only XML output.
JSON_NOT_SUPPORTED_ERR	501	JSON not supported due to a potential large amount of output.
MSG_TYPE_UNSUPPORTED_ERR	501	Message type not supported.
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	Pipe XML for this command is not allowed in input.
STRUCT_NOT_SUPPORTED_ERR	501	Structured output unsupported.
ERR_UNDEFINED	600	Unknown error.