NX-API

This chapter contains the following sections:

About NX-API

NX-API is an enhancement to the Cisco MDS 9000 Series CLI system. NX-API improves the accessibility of the CLIs that are run on the Cisco MDS 9000 devices by making them available outside the switch by using HTTP or HTTPS. CLIs are encoded into the HTTP or HTTPS POST body. 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.

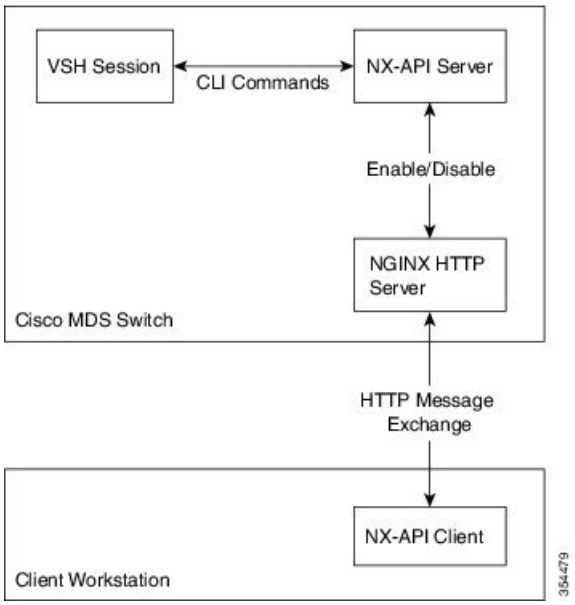
NX-API supports XML, JSON, and JSON-RPC formats for commands and their outputs.

You can use any REST-based tool to interact with a Cisco MDS device. You can also use your own web-based mobile tool that supports sending and receiving HTTP or HTTPS requests and responses to interact with the device.

NX-API Workflow

The NX-API backend uses the NGINX HTTP server. The NGINX server looks out for requests on the HTTP port. Note that HTTPS support is not enabled by default. It has to be enabled by NX-API CLIs.

Figure 1: NX-API Workflow



NX-API Performance

NX-API 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.
- Device (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.
- Parsed and unparsed output of **show** commands—NX-API performance is better with unparsed outputs.

Message Format

- NX-API output presents information in a user-friendly format.
- NX-API does not map directly to the Cisco NX-OS NETCONF implementation.
- NX-API output of supported commands can be viewed in XML, JSON, and JSON-RPC formats.

Security

By default, NX-API uses HTTP basic authentication. 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.

In Cisco NX-OS Release 8.1(x), and 8.2(x), when NX-API is enabled over HTTPS, a 2K 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. In Cisco NX-OS Release 8.3(x), the self-signed certificate expires after 24 hours. We recommend that you use a CA-signed certificate for this release too.

NX-API is integrated into the authentication system of the supported Cisco MDS switches. Users must have appropriate accounts (network-admin, network-operator, and so on) to access a switch 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 device.

Supported Switches

NX-API is supported on the following Cisco MDS 9000 Series Switches:

- Cisco MDS 9700 Series Switches
- Cisco MDS 9250i Multiservice Fabric Switch
- Cisco MDS 9396T Multilayer Fabric Switch
- Cisco MDS 9396S Multilayer Fabric Switch
- Cisco MDS 9148T Multilayer Fabric Switch
- Cisco MDS 9148S Multilayer Fabric Switch

Limitations

- The XML output of the commands listed below will not be supported if the interface type (Fibre Channel or Ethernet) is not explicitly specified. The XML output of these commands is supported only if a Fibre Channel or Ethernet interface is specified in the command. For example, **show interface** *ifid* **bbcredit**.
  - **show interface**
  - **show interface bbcredit**
  - **show interface brief**
  - **show interface capabilities**
  - **show interface counters**
  - **show interface debounce**
  - **show interface description**
  - **show interface detail-counters**
- The XML outputs of FCIP interface related commands are not supported.

Note For more information on platforms supported in Cisco MDS NX-OS Release 8.x, see the Cisco MDS 9000 Series Compatibility Matrix, Release 8.x.

Using NX-API

The commands, command type, and output type for the Cisco MDS 9000 devices should be entered using NX-API by encoding the parameters into the body of an HTTP or HTTPs POST. The response to a request is returned in XML, JSON, or JSON-RPC output format.

NX-API Developer Sandbox

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

By default, NX-API is disabled. Enable NX-API with the **feature nxapi** command in global configuration mode on the corresponding Cisco MDS 9000 switch. Then, enable the NX-API Developer Sandbox using the **nxapi sandbox** command.

The following are the default ports for NX-API:

- HTTP: 8080
- HTTPS
  - 443—for Cisco MDS NX-OS Release 8.2(1) and earlier

HTTP is enabled by default when the **feature nxapi** command is entered. HTTPS is not enabled by default and can be enabled using the command shown in the following example.

The following example shows how to enable NX-API, NX-API Developer Sandbox, and how to enable HTTPS with the relevant port for Cisco MDS NX-OS Release 8.2(1) and earlier:

```
switch# configure terminal
switch(config)# feature nxapi
switch(config)# nxapi sandbox
switch(config)# nxapi https port 443
```

The following sample outputs of the **show nxapi** command displays the status of NX-API, NX-API Developer Sandbox, and the HTTP and HTTPS ports:

Cisco MDS NX-OS Release 8.2(1) and earlier

```
switch# show nxapi
NX-API:       Enabled         Sandbox:      Enabled
HTTP Port:    8080            HTTPS Port:   443
```

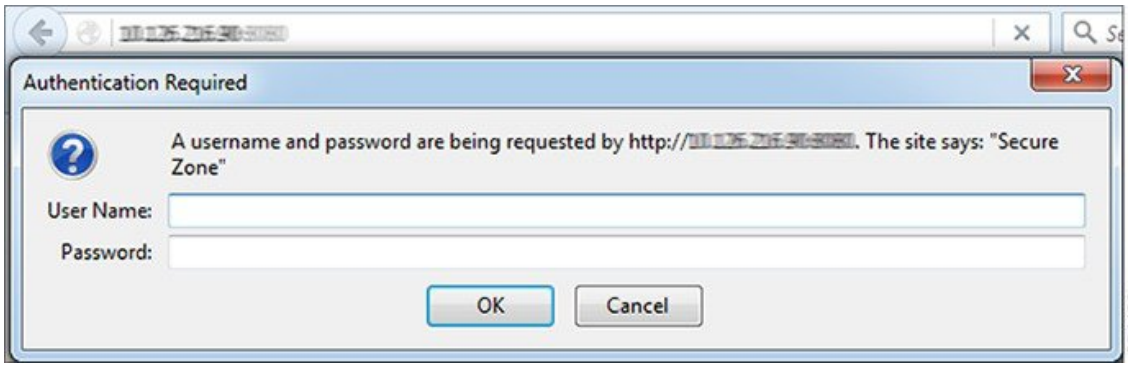
Cisco MDS NX-OS Release 8.3(1) and later

To launch 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* (HTTP) or https://*switch_ip_address*:*port-number* (HTTPS) in the address bar. The following authentication window is displayed:

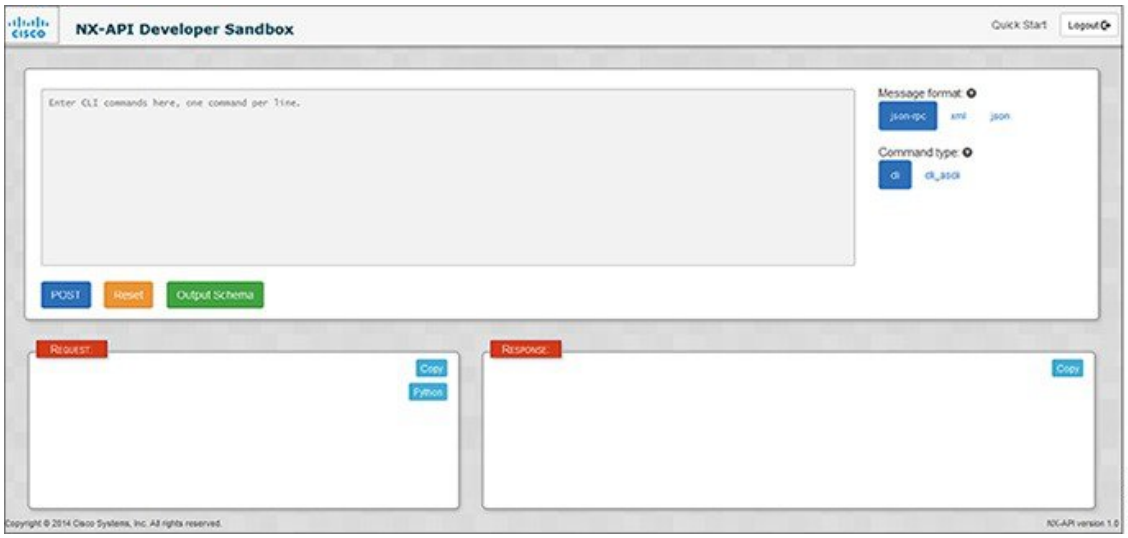
Figure 2: NX-API Developer Sandbox Authentication



2. Log in using your switch credentials.

   The NX-API Developer Sandbox window is displayed.

Figure 3: NX-API Developer Sandbox



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.)

Note

- The **bash** Command type is not supported in Cisco MDS NX-OS Releases 7.x and 8.x.
- If you select the **xml** Message format, **cli_show** and **cli_show_ascii** Command types have the option to enable the chunk mode. 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. Enter the command in the space provided (1, in figure below).
4. The command that you entered is displayed in the selected Message format in the **REQUEST** area (7, in figure below).
   - To copy the data populated in the **REQUEST** area, click **Copy**.
   - To generate a Python code for the command entered, click **Python**.

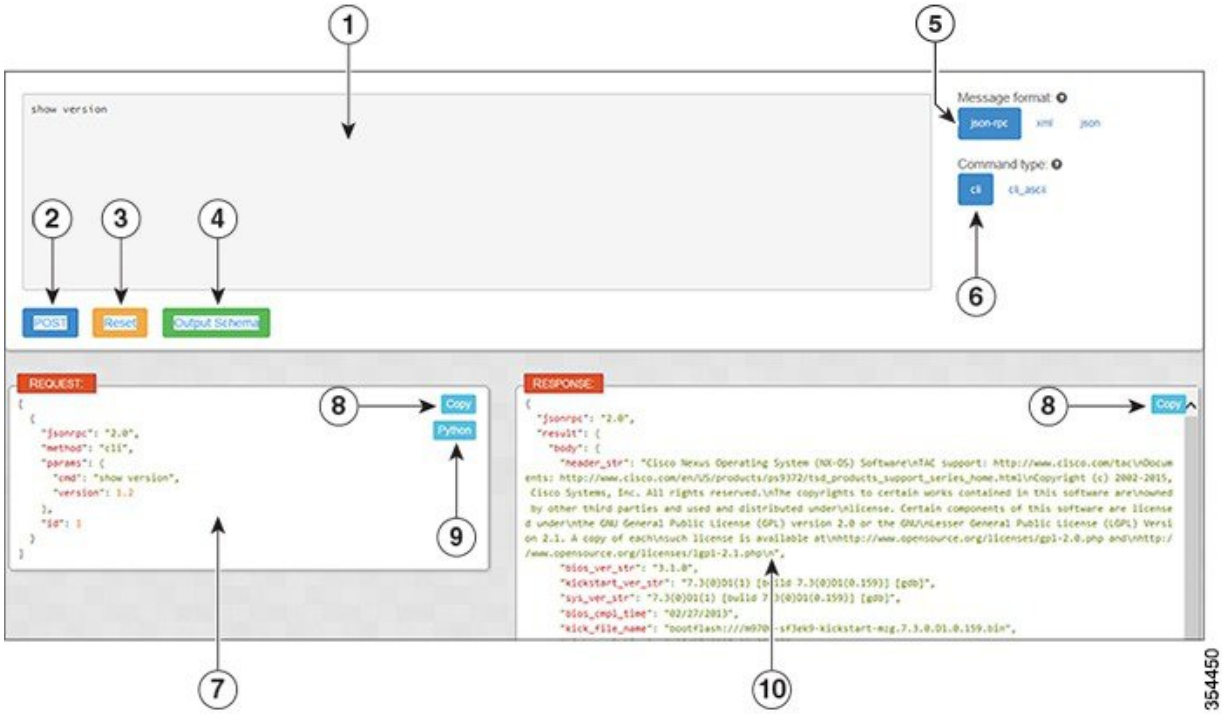
   Note The **xml** Message format does not support the **Python** button.

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

   The output of the command is displayed in the **RESPONSE** area (10, in figure below).
   - 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**.

Figure 4: NX-API Developer Sandbox with Example Request and Output Response



1	Command entry	6	Command type
2	POST	7	REQUEST
3	Reset	8	Copy
4	Output Schema	9	Python
5	Message format	10	RESPONSE

Example: Displaying NX-API Status

The following example displays the NX-API status:

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>
```

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

command | xml

If the command is NX-API-aware, 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:nf="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns="http://www.cisco.com/nxos::ddas">
 <nf:data>
  <show>
   <device-alias>
    <merge>
     <status>
      <__readonly__>
       <result>None</result>
       <reason>None</reason>
      </__readonly__>
     </status>
    </merge>
   </device-alias>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
switch#
```

Note Even if a **show** command is not NX-API-aware, you can still view the output in ASCII format on the NX-API Developer Sandbox by selecting the **cli_show_ascii** command type on the interface.

NX-API Request Elements

NX-API request elements (7, in Figure 4: NX-API Developer Sandbox with Example Request and Output Response) 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 1: NX-API Request Elements

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_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_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.
chunk	Some **show** commands, for example, **show zone**, can return a large amount of output. For an NX-API client to start processing the output before the entire command completes, NX-API supports output chunking for **show** commands. **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.

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 headers indicate the type of response format (XML, JSON, or JSON-RPC).

NX-API Response Elements

The NX-API elements (10, in Figure 4: NX-API Developer Sandbox with Example Request and Output Response) that respond to a CLI command are listed in the following table:

Figure 5: NX-API Response Elements

Table 2: 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 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 3: 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.

NX-API Management Commands

You can enable and manage NX-API on your Cisco MDS switch with the commands listed in the following table.

Table 4: 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-number*	Specifies a port.
no nxapi {**http** | **https**}	Disables HTTP or HTTPS.
show nxapi	Displays port information.
nxapi certificate *certpath* **key** *keypath*	Specifies the upload of the following: • HTTPS certificate when *certpath* is specified. • HTTPS key when *keypath* is specified.