NX-API

NX-APIについて

機能 NX-API

- サンドボックスを介してデバイスにアクセスするには、機能 NX-API を有効にする必要があります。
- デバイス上の | json は、内部的に Python スクリプトを使用して出力を生成します。
- NX-API は、ipv4 を介して http / https のいずれかで有効にすることができます。

```
BLR-VXLAN-NPT-CR-179# show nxapi
 nxapi enabled
 HTTP Listen on port 80
 HTTPS Listen on port 443
 BLR-VXLAN-NPT-CR-179#
```

- NX-API は、サードパーティの NGINX プロセスを内部的に生成しています。このプロセスは、ハンドラ受信 / 送信 / http 要求 の処理 / 応答：

```
nxapi certificate {httpscrt |httpskey}
nxapi certificate enable
```

- NX-API 証明書は https で有効にできます
- nginx が動作するデフォルトのポートは、http / https がそれぞれ 80 / 443 です。次の CLI コマンドを使用して変更することもできます：

```
nxapi {http|https} port port-number
```

転送

NX-APIは、転送のようにHTTPまたはHTTPSを使用します。CLIは、HTTP / HTTPS POST本文にエンコードされます。

NX-APIバックエンドはNginx HTTPサーバを使用します。Nginxプロセスとそのすべての子プロセスは、CPUとメモリの使用量が制限されているLinux cgroup保護下にあります。Nginxのメモリ使用量がcgroupの制限を超えると、Nginxプロセスが再起動されて復元されます。

メッセージ形式

（注）
- NX-API XML出力は、情報を使いやすいフォーマットで表示します。
- NX-API XMLは、Cisco NX-OS NETCONF導入に直接マッピングされません。
- NX-API XML出力は、JSONまたはJSON-RPCに変換できます。

セキュリティ

NX-APIはHTTPSをサポートします。HTTPSを使用すると、デバイスへのすべての通信が暗号化されます。

NX-APIは、デバイスの認証システムに統合されています。ユーザーは、NX-APIを介してデバイスにアクセスするための適切なアカウントを持っている必要があります。NX-APIではHTTP basic認証が使用されます。すべてのリクエストには、HTTPヘッダーにユーザー名とパスワードが含まれている必要があります。

（注） ユーザーのログイン資格情報を保護するには、HTTPSの使用を検討する必要があります。

[機能（feature）]マネージャCLIコマンドを使用して、NX-APIを有効にすることができます。NX-APIはデフォルトで無効になっています。

NX-APIの使用

デバイスで**feature** manager CLIコマンドを使用してNX-APIを有効にする必要があります。デフォルトでは、NX-APIは無効になっています。

次の例は、NX-APIサンドボックスを設定して起動する方法を示しています。

- 管理インターフェイスを有効にします。

```
switch# conf t
switch(config)# interface mgmt 0
```

```
switch(config)# ip address 198.51.100.1/24
switch(config)# vrf context managment
switch(config)# ip route 203.0.113.1/0 1.2.3.1
```

- NX-API **nxapi** 機能を有効にします。

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

次の例は、リクエストとそのレスポンスを XML 形式で示しています。

要求:

```
<?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>
```

応答：

```
<?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>
```

次の例は、JSON 形式の要求とその応答を示しています。

要求:

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

応答：

```
{
    "ins_api": {
        "type": "cli_show",
```

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

NX-API コールの管理インターフェイスの使用

NX-API コールには管理インターフェイスを使用することをお勧めします。

NX-API の非管理インターフェイスとカスタム ポートを使用する場合、NX-API トラフィックが API トラフィックを好ましくない処理する可能性のあるデフォルトの copp エントリにヒットしないように、CoPP ポリシーにエントリを作成する必要があります。

（注） NX-API トラフィックには管理インターフェイスを使用することをお勧めします。それが不可能で、カスタム ポートが使用されている場合は、「copp-http」クラスを更新して、カスタム NX-API ポートを含める必要があります。

次の例のポート 9443 は、NX-API トラフィックに使用されています。

このポートは、copp-system-acl-http ACL に追加され、copp-http クラスの下で一致できるようになり、100 pps ポリシングになります。（特定の環境では、これを増やす必要がある場合があります。）

```
!
ip access-list copp-system-acl-http
  10 permit tcp any any eq www
  20 permit tcp any any eq 443
  30 permit tcp any any eq 9443 <---------
!
class-map type control-plane match-any copp-http
  match access-group name copp-system-acl-http
!
!
policy-map type control-plane copp-system-policy
  class copp-http
    police pps 100
!
```

NX-API 管理コマンド

次の表にリストされている CLI コマンドを使用して、NX-API を有効にして管理できます。

表 *1 : NX-API* 管理コマンド

NX-API 管理コマンド	説明
feature nxapi	NX-API を有効化します。
no feature nxapi	NX-API を無効化します。
nxapi {**http** ｜ **https**} **port** *port*	ポートを指定します。
no nxapi {**http** ｜ **https**}	HTTP / HTTPS を無効化します。
show nxapi	ポート情報を表示します。
nxapi certificate {httpscrt certfile ｜ **httpskey keyfile**} *filename*	次のアップロードを指定します： • httpscrt が指定されている場合の HTTPS 証明書。 • httpskey が指定されている場合の HTTPS キー。 HTTPS 証明書の例： **nxapi certificate httpscrt certfile bootflash:cert.crt** HTTPS キーの例： **nxapi certificate httpskey keyfile bootflash:privkey.key**
nxapi certificate enable	証明書を有効化します。

以下は、HTTPS 証明書の正常なアップロードの例です：

```
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)#
```

以下は、HTTPS キーの正常なアップロードの例です：

```
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)#
```

状況によっては、証明書が無効であることを示すエラーメッセージが表示されることがあります：

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

これは、キーファイルが暗号化されている場合に発生する可能性があります。その場合、キーファイルをインストールする前に復号化する必要があります。次の例に示すように、Guest Shell に移動してキー ファイルを復号化する必要がある場合があります。

```
switch(config)# guestshell
[b3456@guestshell ~]$
[b3456@guestshell bootflash]$ /bin/openssl rsa -in certfilename.net.pem -out clearkey.pem

Enter pass phrase for certfilename.net.pem:
writing RSA key
[b3456@guestshell bootflash]$
[b3456@guestshell bootflash]$ exit
switch(config)#
```

これが問題の原因である場合、証明書を正常にインストールできるはずです。

```
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)#
```

NX-API を使用したインタラクティブ コマンドの操作

対話型コマンドの確認プロンプトを無効にし、エラー コード 500 によるタイムアウトを回避するには、対話型コマンドの前に**[端末の dont-ask（terminal dont-ask）]** を追加します。 **を使用。複数の対話型コマンドを区切るには**、それぞれが **。**は単一のブランク文字で囲まれています。

エラー コード 500 でのタイムアウトを回避するために**端末の dont-ask** を使用する対話型コマンドの例をいくつか次に示します：

```
terminal dont-ask ; reload module 21

terminal dont-ask ; system mode maintenance
```

NX-API リクエスト要素

NX-API 応答要素

CLI コマンドに応答する NX-API 要素を次の表に示します。

表 *2: NX-API* 応答要素

NX-API 応答要素	**説明**
version	NX-API バージョン。
type	実行するコマンドのタイプ。
sid	応答のセッション識別子。この要素は、応答メッセージがチャンクされている場合にのみ有効です。

NX-API 応答要素	説明
outputs	すべてのコマンド出力を囲むタグ。 複数のコマンドが cli_show または cli_show_ascii にある場合、各コマンド出力は単一の出力タグで囲まれます。 メッセージ タイプが cli_conf または bash の場合、cli_conf および bash コマンドにはコンテキストが必要なため、すべてのコマンドに単一の出力タグがあります。
出力	単一のコマンド出力の出力を囲むタグ。 cli_conf と bash メッセージ タイプの場合、この要素にはすべてのコマンドの出力が含まれます。
input	リクエストで指定された 1 つのコマンドを囲むタグ。この要素は、要求入力要素を適切な応答出力要素に関連付けるのに役立ちます。
本文	コマンド応答の本文。
コード	コマンドの実行から返された原因コード。 NX-API は、ハイパーテキスト転送プロトコル（HTTP）ステータス コード レジストリ（http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml）で説明されている標準規格の HTTP原因コードを使用します。
msg	返された原因コードに関連付けられたエラー メッセージ。

JSON の概要（JavaScript オブジェクト表記）

JSONは、判読可能なデータのために設計された軽量テキストベースのオープン スタンダードで、XML の代替になります。JSON はもともと JavaScript から設計されましたが、言語に依存しないデータ形式です。JSON/CLI 実行は現在、Cisco Nexus 3500 プラットフォーム スイッチでサポートされています。

（注） NX-API/JSON 機能は、Cisco Nexus 3500 プラットフォーム スイッチで使用できるようになりました。

ほぼすべての最新のプログラミング言語で何らかの方法でサポートされている 2 つの主要なデータ構造は次のとおりです。

- 順序付きリスト :: 配列
- 順序付けられていないリスト（名前/値のペア）:: オブジェクト

show コマンドの JSON/JSON-RPC/XML 出力には、サンドボックス経由でアクセスすることもできます。

CLI の実行

Show_Command | json

コード例

```
BLR-VXLAN-NPT-CR-179# show cdp neighbors  | json
{"TABLE_cdp_neighbor_brief_info": {"ROW_cdp_neighbor_brief_info": [{"ifindex": "
83886080", "device_id": "SW-SPARSHA-SAVBU-F10", "intf_id": "mgmt0", "ttl": "148"
, "capability": ["switch", "IGMP_cnd_filtering"], "platform_id": "cisco WS-C2960
S-48TS-L", "port_id": "GigabitEthernet1/0/24"}, {"ifindex": "436207616", "device
_id": "BLR-VXLAN-NPT-CR-178(FOC1745R01W)", "intf_id": "Ethernet1/1", "ttl": "166
", "capability": ["router", "switch", "IGMP_cnd_filtering", "Supports-STP-Disput
e"], "platform_id": "N3K-C3132Q-40G", "port_id": "Ethernet1/1"}]}}
BLR-VXLAN-NPT-CR-179#
```

XML および JSON でサポートされたコマンド

NX-OS は、次の構造化された出力フォーマットで、さまざまな **show** コマンドの標準規格出力のリダイレクトをサポートしています。

- XML
- JSON
- JSON フォーマット出力の標準規格ブロックを読みやすくする JSON Pretty

標準規格の NX-OS 出力を JSON、JSON Pretty、または XML フォーマットに変換することは、出力を JSON または XML インタープリターに「パイプ」することによって、NX-OS CLI で発生します。たとえば、論理パイプ (|) を使用して **show ip access** コマンドを発行し、JSON、JSON Pretty、または XML を指定すると、NX-OS コマンド出力が適切に構造化され、そのフォーマットでエンコードされます。この機能により、プログラムによるデータの解析が可能になり、ソフトウェア ストリーミング テレメトリを介したスイッチからのストリーミング データがサポートされます。Cisco NX-OS のほとんどのコマンドは、JSON、JSON Pretty、および XML 出力をサポートしています。

この機能の選択された例を以下に表示します。

XML および JSON 出力の例

次の例は、ハードウェア テーブルのユニキャストおよびマルチキャスト ルーティング エントリを JSON 形式で表示する方法を示しています。

```
switch(config)# show hardware profile status | json
{"total_lpm": ["8191", "1024"], "total_host": "8192", "max_host4_limit": "4096",
 "max_host6_limit": "2048", "max_mcast_limit": "2048", "used_lpm_total": "9", "u
sed_v4_lpm": "6", "used_v6_lpm": "3", "used_v6_lpm_128": "1", "used_host_lpm_tot
al": "0", "used_host_v4_lpm": "0", "used_host_v6_lpm": "0", "used_mcast": "0", "
used_mcast_oifl": "2", "used_host_in_host_total": "13", "used_host4_in_host": "1
2", "used_host6_in_host": "1", "max_ecmp_table_limit": "64", "used_ecmp_table":
```

```
"0", "mfib_fd_status": "Disabled", "mfib_fd_maxroute": "0", "mfib_fd_count": "0"
}
switch(config)#
```

次に、ハードウェア テーブルのユニキャストおよびマルチキャスト ルーティング エントリを XML 形式で表示する例を示します。

```
switch(config)# show hardware profile status | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:fib">
 <nf:data>
  <show>
   <hardware>
    <profile>
     <status>
      <__XML__OPT_Cmd_dynamic_tcam_status>
       <__XML__OPT_Cmd_dynamic_tcam_status___readonly__>
        <__readonly__>
         <total_lpm>8191</total_lpm>
         <total_host>8192</total_host>
         <total_lpm>1024</total_lpm>
         <max_host4_limit>4096</max_host4_limit>
         <max_host6_limit>2048</max_host6_limit>
         <max_mcast_limit>2048</max_mcast_limit>
         <used_lpm_total>9</used_lpm_total>
         <used_v4_lpm>6</used_v4_lpm>
         <used_v6_lpm>3</used_v6_lpm>
         <used_v6_lpm_128>1</used_v6_lpm_128>
         <used_host_lpm_total>0</used_host_lpm_total>
         <used_host_v4_lpm>0</used_host_v4_lpm>
         <used_host_v6_lpm>0</used_host_v6_lpm>
         <used_mcast>0</used_mcast>
         <used_mcast_oifl>2</used_mcast_oifl>
         <used_host_in_host_total>13</used_host_in_host_total>
         <used_host4_in_host>12</used_host4_in_host>
         <used_host6_in_host>1</used_host6_in_host>
         <max_ecmp_table_limit>64</max_ecmp_table_limit>
         <used_ecmp_table>0</used_ecmp_table>
         <mfib_fd_status>Disabled</mfib_fd_status>
         <mfib_fd_maxroute>0</mfib_fd_maxroute>
         <mfib_fd_count>0</mfib_fd_count>
        </__readonly__>
       </__XML__OPT_Cmd_dynamic_tcam_status___readonly__>
      </__XML__OPT_Cmd_dynamic_tcam_status>
     </status>
    </profile>
   </hardware>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

この例では、JSON 形式でスイッチ上に LLDP タイマーを表示する例を示します。

```
switch(config)# show lldp timers | json
{"ttl": "120", "reinit": "2", "tx_interval": "30", "tx_delay": "2", "hold_mplier
": "4", "notification_interval": "5"}
```

```
switch(config)#
```

この例では、XML 形式でスイッチ上に LLDP タイマーを表示する例を示します。

```
switch(config)# show lldp timers | xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://w
ww.cisco.com/nxos:1.0:lldp">
 <nf:data>
  <show>
   <lldp>
    <timers>
     <__XML__OPT_Cmd_lldp_show_timers___readonly__>
      <__readonly__>
       <ttl>120</ttl>
       <reinit>2</reinit>
       <tx_interval>30</tx_interval>
       <tx_delay>2</tx_delay>
       <hold_mplier>4</hold_mplier>
       <notification_interval>5</notification_interval>
      </__readonly__>
     </__XML__OPT_Cmd_lldp_show_timers___readonly__>
    </timers>
   </lldp>
  </show>
 </nf:data>
</nf:rpc-reply>
]]>]]>
switch(config)#
```

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。