NX-API CLI

NX-API CLI について

NX-API CLI は、XML 出力をサポートする Cisco NX-OS CLI システムの拡張機能です。NX-API CLI は、特定のコマンドの JSON 出力フォーマットもサポートしています。

Cisco Nexus スイッチでは、コマンドライン インターフェイス（CLI）はスイッチ上でのみ実行されます。NX-API CLI は HTTP / HTTPS を使ってスイッチの外部で CLI を使用できるようにすることで、これらの CLI のユーザー補助を改善します。この拡張機能をスイッチの既存の Cisco NX-OS CLI システムに使用できます。NX-API CLI は **show** コマンド、構成と Linux Bash をサポートします。

NX-API CLI は JSON-RPC をサポートしています。

注意事項と制約事項

NX-API CLI は、スイッチで Cisco NX-OS CLI を実行するために VSH を生成します。VSH のタイムアウトは 5 分です。Cisco NX-OS CLI の実行に 5 分以上かかると、コマンドは失敗し、「Back-end processing error.」というメッセージが表示されます。これは、NX-API コマンドのタイムアウトによって制御されます。これは、NX-API を介して要求されたコマンドを実行できる時間を制御します。この値は 300 秒に固定されており、変更できません。

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。

Cisco NX-OS リリース 9.2(1) 以降、NX-API 機能は HTTPS ポート 443 でデフォルトで有効になっています。HTTP ポート 80 は無効です。

NX-APIは、ホスト上でネイティブに、またはゲストシェル内で実行されるアプリケーションの、UNIX ドメイン ソケットを介してサポートされます。

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。NX-API プロセスは、cgroup ext_ser_nginx の一部であり、2,147,483,648 バイトのメモリに制限されています。Nginx のメモリ使用量が cgroup の制限を超えると、Nginx プロセスは再起動されて、NX-API 構成（VRF、ポート、証明書構成）が復元されます。

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 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、デフォルトでは非セキュア HTTP をサポートしていません。
- NX-API は、デフォルトでは弱い TLSv1 プロトコルをサポートしていません。

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 は、ユーザーが最初に認証に成功したときに、セッション ベースの Cookie、**nxapi_auth** を提供します。セッション Cookie を使用すると、デバイスに送信される後続のすべての NX-API 要求にユーザー名とパスワードが含まれます。ユーザー名とパスワードは、完全な認証プロセスの再実行をバイパスするために、セッション Cookie で使用されます。セッション Cookie が後続の要求に含まれていない場合は、別のセッション Cookie が必要であり、認証プロセスによって提

供されます。認証プロセスの不必要な使用を避けることで、デバイスのワークロードを軽減できます。

（注） **nxapi_auth** cookie は 600 秒（10 分）で期限切れになります。この値は固定されており、調整できません。

（注） NX-API は、スイッチ上の Programmable Authentication Module（PAM）を使用して認証を行います。cookie を使用して PAM の認証数を減らし、PAM の負荷を減らします。

NX-API CLI の使用

Cisco Nexus 9000 シリーズ スイッチのコマンド、コマンド タイプ、および出力タイプは、CLI を HTTP/HTTPS POST の本文にエンコードすることにより、NX-API を使用して入力されます。要求に対する応答は、XML または JSON 出力形式で返されます。

（注） NX-API 応答コードの詳細については、NX-API 応答コードの表 （26 ページ）を参照してください。

NX-API CLI は、ローカル アクセスに対してはデフォルトで有効になっています。リモート HTTP アクセスに対してはデフォルトで無効になっています。

次の例は、NX-API CLI を構成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
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#
```

- 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"
            }
        }
    }
}
```

（注） ユーザーを削除しようとすると失敗し、次のようなエラー メッセージが約 12 時間ごとに表示されるという既知の問題があります。

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

この問題は、NX-APIを介してスイッチにログインしているユーザーを削除しようとした場合に発生する可能性があります。この場合、次のコマンドを入力して、最初にユーザーのログアウトを試行します。

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

その後、ユーザーの削除を再試行します。回避策を試みても問題が解決しない場合は、Cisco TAC へお問い合わせください。

NX-API で権限を root にエスカレーションする

NX-API では、管理者ユーザーの権限を root アクセスの権限にエスカレーションできます。

以下は、権限をエスカレーションするためのガイドラインです：

- 特権を root にエスカレーションできるのは管理者ユーザーのみです。
- root へのエスカレーションはパスワードで保護されています。

次の例は、管理者の権限を root にエスカレーションする方法と、エスカレーションを確認する方法を示しています。root になっても、**whoami** コマンドを実行すると admin として表示されることに注意してください。ただし、admin アカウントにはすべての root 権限があります。

最初の例：

```
<?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>
```

2 番目の例：

```
<?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>
```

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	ポートと証明書情報を表示します。 （注） 「**show nxapi**」コマンドは、network-operator ロールの証明書/構成情報を表示しません。

NX-API 管理コマンド	説明
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 certificatehttpskey keyfile *filename* **password** *passphrase*	暗号化された秘密キーを使用してNX-API証明書をインストールします。 （注） 暗号化された秘密キーを復号するためのパスフレーズは **pass123!** です。 例: **nxapi certificate httpskey keyfile bootflash:encr-cc.pem password pass123!**
nxapi certificate enable	証明書を有効化します。

NX-API 管理コマンド	説明
nxapi certificate sudi	この CLI は、Secure Unique Device Identifier（SUDI）を使用してデバイスを安全に認証する方法を提供します。 nginx の SUDI ベースの認証は、CISCO SUDI 準拠のコントローラによって使用されます。 SUDI は、X.509v3 証明書に含まれる IEEE 802.1AR 準拠のセキュアデバイス識別子で、Cisco デバイスの製品識別子とシリアル番号を維持します。ID は製造時に実装され、公的に識別可能なルート認証局につながれます。 （注） NX-API が SUDI 証明書を使用している場合、ブラウザ、curl などのサードパーティ アプリケーションからはアクセスできません。 （注） 「nxapi certificate sudi」は、構成されるとカスタム証明書/キーを上書きします。カスタム証明書/キーを元に戻す方法はありません。 （注） 「nxapi certificate sudi」と「nxapi certificate trustpoint」と「nxapi certificate enable」は排他的であり、いずれかを構成すると他の構成は削除されます。 （注） NX-API は、SUDI 証明書ベースのクライアント証明書認証をサポートしていません。クライアント証明書認証が必要な場合は、アイデンティティ証明書を使用する必要があります。 （注） NX-API 証明書 CLI は show run の出力に存在しないため、現在の CR/ロールバックでは、「nxapi certificate sudi」オプションで上書きした場合、カスタム証明書に戻せません。
nxapi ssl-ciphers weak	Cisco NX-OS リリース 9.2(1) 以降、弱い暗号はデフォルトで無効になっています。このコマンドを実行すると、デフォルトの動作が変更され、NGINX の弱い暗号が有効になります。このコマンドの **no** 形式を使用すると、デフォルトに戻ります（デフォルトでは弱い暗号は無効です）。

NX-API 管理コマンド	説明
nxapi ssl-protocols {TLSv1.0 TLSv1.1 TLSv1.2}	Cisco NX-OS リリース 9.2(1) 以降、TLS1.0 はデフォルトで無効になっています。このコマンドを実行すると、文字列で指定された TLS バージョンが有効になります。デフォルトで無効になっている TLS1.0 も、必要であれば指定できます。このコマンドの **no** 形式を使用すると、デフォルトに戻ります（デフォルトでは、TLS1.1 と TLS1.2 のみが有効です）。
nxapi use-vrf *vrf*	デフォルト VRF、管理 VRF、または名前付き VRF を指定します。
ip netns exec management iptables	アクセス制限を実装したうえで、管理 VRF で実行できます。 （注） **feature bash-shell** を有効にしてから、Bash シェルからコマンドを実行する必要があります。Bash シェルの詳細については、Bash の章を参照してください。 *Iptables* は、ポリシーチェーンを使用してトラフィックを許可または阻止するコマンドライン ファイアウォール ユーティリティであり、ほとんどの *Linux* ディストリビューションにプリインストールされています。 （注） iptables を bash シェルで変更した後、リロード後も iptables を永続化する方法の詳細については、リロード間で Iptable を永続化する（24 ページ）を参照してください。
nxapi idle-timeout <timeout>	リリース 9.3(5) 以降では、アイドル状態の NX-API セッションが無効になるまでの時間を設定できます。指定できる時間は 1 ～ 1440 分です。デフォルトの時間は 10 分です。デフォルト値に戻すには、**no nxapi idle-timeout <timeout>** コマンドの no 形式を使用します。

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)#
```

（注） 証明書を有効にする前に、証明書とキーを構成する必要があります。

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)#
```

次に、暗号化された NXAPI サーバー証明書をインストールする方法の例を示します。

```
switch(config)# nxapi certificate httpscrt certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key password pass123!

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 httpscrt certfile bootflash:certificate.crt
switch(config)# nxapi certificate httpskey keyfile bootflash:privkey.key
ERROR: Unable to load private key!
Check keyfile or provide pwd if key is encrypted, using 'nxapi certificate httpskey keyfile
 <keyfile> password <passphrase>'.
```

この場合、**nxapi certificatehttpskey keyfile** *filename* **password** *passphrase*を使用して暗号化キー ファイルのパスフレーズを指定する必要があります。

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 password pass123!
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 クライアント認証

NX-API Client Basic Authentication

NX-API clients can authenticate with the NGINX server on the switch through basic authentication over SSL/TLS. This authentication method is supported by configuring a username and password that is saved to a database on the switch. When the NX-API client initiates a connection request, it sends the Hello message which contains the username and password. Assuming the username and password exist in the database, the switch responds by sending the Hello response, which contains a cookie. After this initial handshake is complete, the communication session is open, and the client can begin sending API calls to the switch. For additional information, see セキュリティ, on page 2.

For additional information about basic authentication, including how to configure the username and password on the switch, refer to the Cisco Nexus 9000 Series NX-OS Security Configuration Guide.

NX-API のクライアント証明書認証

NX-OS 9.3(3) 以降、NX-API はクライアントが開始する証明書ベースの認証をサポートしています。証明書ベースの認証では、TLS ハンドシェイク時に信頼できる関係者、つまり認証局（CA）を使用してクライアントとサーバーの両方を相互に認証することで、セキュリティを強化します。証明書ベースの認証では、NX-OS スイッチにアクセスするためのマシン認証だけでなく、人間による認証も可能です。

クライアント証明書認証は、有効な CA（認証局）を介して割り当てられ、NX-API クライアントに保存されている X509 SSL 証明書を使用してサポートされます。証明書は、各 NX-API ユーザー名に割り当てられます。

NX-API クライアントが Hello メッセージを使用して接続要求を開始すると、サーバーの Hello 応答に有効な CA のリストが含められます。クライアントの応答には、NX-API クライアントが使用している特定のユーザー名の証明書など、追加の情報要素が含まれます。

NX-API クライアントは基本認証、証明書認証のいずれかを使用するように構成することができます。または証明書を優先するものの、証明書認証方式が使用できない場合は基本認証にフォールバックするように構成することもできます。

注意事項と制約事項

証明書認証には次の注意事項と制約事項があります。

- NX-API クライアントには、ユーザー名とパスワードを設定する必要があります。
- NX-API クライアントとスイッチは、デフォルトでウェルノウン ポートで HTTP を介して通信します。柔軟性を高めるために、HTTP はウェルノウン ポートでもサポートされます。ただし、追加のポートを設定することもできます。
- クライアント証明書認証の Python スクリプティングがサポートされています。クライアント証明書がパスフレーズで暗号化されている場合、python はパスフレーズの入力を通常どおり要求します。ただし、Python 要求ライブラリの現在の制限により、パスフレーズをスクリプトに渡すことはできません。
- NX-API クライアントとスイッチは、同じトラストポイントを使用する必要があります。
- サポートされるトラストポイントの最大数は、スイッチごとに 26 です。
- 信頼できる CA のリストは、すべての NX-API クライアントとスイッチで同じである必要があります。信頼できる CA の個別のリストはサポートされていません。
- 証明書認証は、NX-API サンドボックスではサポートされていません。
- 次の条件によって、NX-API サンドボックスがスイッチにロードされるかどうかが決まります。
  - NX-API サンドボックスは、**nxapi client certificate authentication optional** または **no nxapi client certificate authentication** が構成されている場合にのみロードされます。

- NX-API サンドボックスは、接続の確立時に有効なクライアント証明書がブラウザに提示されない限り、**strict** および **two-step** 認証モードをロードしません。

- スイッチには NGINX サーバーが組み込まれています。複数のトラストポイントが設定されていても、証明書失効リスト（CRL）が 1 つのトラストポイントだけにインストールされている場合、NGINX の制限により NX-API クライアント証明書認証は失敗します。この制限を回避するには、すべてのトラストポイントに CRL を構成します。
- 証明書は期限切れになったり、期限切れになったりする可能性があり、CA（トラストポイント）によって設定された CRL の有効性に影響を与える可能性があります。スイッチが有効な CRL を使用できるように、設定されているすべてのトラストポイントに必ず CRL をインストールしてください。トラストポイントによって失効した証明書が存在しない場合は、空の CRL を生成、インストールし、更新する必要があります。これはたとえば週に 1 回行います。

  暗号化 CLI を使用して CRL を更新した後、**nxapi client cert authentication** を発行して、新しく更新された CRL を再適用します。
- NX-API クライアント証明書認証が有効になっているときに ASCII リロードを使用する場合は、リロードの完了後に **nxapi client certificate authentication** を発行する必要があります。
- 証明書パスは信頼済み CA 証明書で終了している必要があります。
- TLS 用に提示されるサーバー証明書には、extendedKeyUsage フィールドにサーバー認証目的（OID 1.3.6.1.5.5.7.3.1 の id-kp 1）が必要です。
- TLS 用に提示されるクライアント証明書には、extendedKeyUsage フィールドにサーバー認証目的（OID 1.3.6.1.5.5.7.3.2 の id-kp 1）が必要です。
- この機能は CRL（証明書失効リスト）をサポートします。オンライン証明書ステータス プロトコル（OCSP）はサポートされていません。
- 『NX-OS セキュリティ ガイド』の追加の注意事項と制約事項に従ってください。
  - 証明書と基本認証の両方を使用します。そうすることで、証明書が何らかの理由で侵害された場合でも、正しいユーザーとパスワードが要求されます。
  - 秘密キーは秘密にしておきます。サーバーの公開キーには接続を試みるすべてのユーザーがアクセスできるからです。
  - CRL は中央 CA からダウンロードし、最新の状態に保つ必要があります。古い CRL はセキュリティリスクにつながる可能性があります。
  - トラストポイントは最新の状態に保ちます。トラストポイントまたは構成の変更が証明書認証機能に加えられた場合は、更新された情報をリロードするために、この機能を明示的に無効にしてから再度有効にします。

- **nxapi certificate httpscert certfile bootflash:<>**「CLI」で NX-API に関連付けられるクライアント証明書アイデンティティ ファイルには、最大 8K というファイルサイズの制限があります。これは Day-1 の制限です。

- NX-API 管理コマンドの表 1 において、コマンド nxapi certificate {httpscrt certfile | httpskey keyfile} filename に関連付けられる行で、サポートされる証明書ファイルの最大サイズは 8K 未満です。

NX-API Client Certificate Authentication Prerequisites

Before configuring certificate authentication, make sure the following are present on the switch:

1. Configure the client with a username and password. For information see Configuring User Accounts and RBAC.
2. Configure the CA(s) (trustpoint) and CRL(s) (if any).

If no certificates were revoked by a trustpoint, create a blank CRL for each trustpoint.

For information, see the Cisco Nexus 9000 Series NX-OS Security Configuration Guide.

NX-API クライアント証明書認証の構成

nxapi client certificate authentication コマンドを使用して、NX-API 証明書認証を設定できます。コマンドは、認証方法を制御する制限オプションをサポートします。

この機能は、**no nxapi client certificate authentication** を使用して無効にすることができます。

NX-API クライアントの証明書認証を構成するには、次の手順を実行します。

手順の概要

1. この機能の前提条件が満たされていることを確認します。
2. **config terminal**
3. **nxapi client certificate authentication [{optional | strict | two-step}]**

手順の詳細

	コマンドまたはアクション	目的
Step 1	この機能の前提条件が満たされていることを確認します。	「NX-API Client Certificate Authentication Prerequisites（13 ページ）」を参照してください。

Step 2	**config terminal**
	例：
	`switch-1# config terminal`
	`Enter configuration commands, one per line. End with CNTL/Z.`
	`switch-1(config)#`
		コンフィギュレーション モードに入ります。

Step 3	**nxapi client certificate authentication [{optional | strict | two-step}]** 例：	次のいずれかのモードで証明書認証を有効にします。 • **optional** はクライアント証明書を要求します。

	コマンドまたはアクション	目的
	`switch-1# nxapi client certificate authentication strict` `switch-1(config)#`	• クライアントが証明書を提供すると、クライアントとサーバーの間で相互検証が行われます。 • クライアントが無効な証明書を提供した場合、認証は失敗し、基本認証へのフォールバックは行われません。 • クライアントが証明書を提供しなかった場合、認証は基本認証（ユーザー名とパスワード）にフォールバックします。 • **strict** は、クライアント証明書の検証を有効にし、認証のために有効なクライアント証明書の提示を要求します。 • **two-step** は、基本認証方式と証明書認証方式の両方が必要な 2 段階検証を有効にします。 （注） スイッチにトラストポイントが構成されていない場合は、この機能を有効にすることはできず、スイッチの画面にエラーメッセージが表示されます。 `No trustpoints configured! Please configure trustpoint using 'crypto ca trustpoint <trustpoint-label>' and associated commands, and then enable this feature.`

証明書認証用の Python スクリプトの例

次の例は、認証用のクライアント証明書を使用した Python スクリプトを示しています。

```
import requests
import json

"""
Modify these please
"""
switchuser='USERID'
switchpassword='PASSWORD'
mgmtip='NXOS MANAGEMENT IP/DOMAIN NAME'

client_cert_file='PATH_TO_CLIENT_CERTIFICATE'
client_key_file='PATH_TO_CLIENT_KEY_FILE'
ca_cert='PATH_TO_CA_CERT_THAT_SIGNED_NXAPI_SERVER_CERT'

url='https://' + mgmtip + '/ins'
myheaders={'content-type':'application/json-rpc'}
payload=[
  {
    "jsonrpc": "2.0",
```

```
    "method": "cli",
    "params": {
      "cmd": "show clock",
      "version": 1
    },
    "id": 1
  }
]
response = requests.post(url,data=json.dumps(payload),
headers=myheaders,auth=(switchuser,switchpassword),cert=(client_cert_file_path,client_key_file),verify=ca_cert).json()
```

必要に応じて、スクリプトを変更できます。

- クライアント証明書認証モードによっては、スイッチ パスワードをヌル値に設定することで（`switchpassword=` ）、スイッチ パスワードを省略できます。
  - **optional** および **strict** モードの場合、`switchpassword=` は空白のままにできます。この場合、NX-APIはユーザー名とクライアント証明書のみに基づいてクライアントを認証します。
  - **two-step** モードの場合、パスワードが必要なため、`switchpassword=` の値を指定する必要があります。
- POST コマンドで `verify=False` を設定することで、NX-API サーバの証明書が有効であることの確認をバイパスできます。

cURL 証明書要求の例

次に、NX-API クライアント認証用の正しく構造化された cURL 証明書要求の例を示します。

```
/usr/bin/curl --user admin: --tlsv1.2 --cacert ./ca.pem --cert ./user.crt:pass123! --key
./user.key -v -X POST -H "Accept: application/json" -H "Content-type: application/json"
--data '{"ins_api":{"version": "1.0", "type": "cli_show", "chunk": "0", "sid": "1", "input":
 "show clock","output_format": "json"}}' https://<device-management-ip>:443/ins
```

構文要素

次の表は、この要求で使用されるパラメータを示しています。

パラメータ	説明
--user	ユーザーがログインするユーザー名を取得します。これは、user.crt の共通名と同じである必要があります。 ユーザーのパスワードを指定するには、コロンの後にパスワードを指定します。例：**--user username:password**

パラメータ	説明
--cacert	NX-API サーバー証明書に署名した CA へのパスを使用します。 サーバー証明書を検証する必要がない場合は、**-k**（insecure）オプションを使用して cURL を指定します。例：**`/usr/bin/curl -k`**
--cert	クライアント証明書へのパスを指定します。 クライアント証明書が暗号化されている場合は、コロンの後にパスワードを指定します。例：**`--cert user.crt:pass123!`**
--key	クライアント証明書の秘密キーへのパスを指定します。

証明書認証の検証

正しく構成されている場合、証明書認証が行われ、NX-API クライアントはスイッチにアクセスできます。

NX-API クライアントがスイッチにアクセスできない場合は、次のガイドラインに従ってトラブルシューティングを行うことができます。

手順の概要

1. ユーザーまたはクッキーのエラーを確認します。
2. 証明書に誤りがないか確認してください。
3. エラーが発生した場合は、 **no nxapi client certificate authentication** 、それから **nxapi client certificate authentication** を発行して、トラストポイント、CA、CRL、または NX-OS 証明書機能に対する変更をリロードするように機能をフラップします。

手順の詳細

	コマンドまたはアクション	目的
Step 1	ユーザーまたはクッキーのエラーを確認します。	次のいずれかのエラーが発生していないか、確認します： • 認証ヘッダーにユーザー名が指定されておらず、有効なクッキーが指定されていない • 認証ヘッダーで指定されたユーザーが正しくない • 無効なクッキーが提供された

	コマンドまたはアクション	目的

		• 認証ヘッダーのユーザー名とクライアント証明書の CN フィールドのユーザー名が一致しない
		
		使用されている NX-API 方式に応じて、特定のエラーが表示されます：
		• JSON/XML の場合、401 認証エラー：ユーザーが見つからないエラーが発生します。次に例を示します。
		`{{{`
		`"code": "400",`
		`"msg": "Authentication failure - user not found."`
		`}}}`
		• JSON RPC 2.0 の場合、-32004 無効なユーザー名またはパスワードエラーが発生します。次に例を示します。
		`{{`
		`"code": -32004,`
		`"message": "Invalid username or password"`
		`}}`

Step 2	証明書に誤りがないか確認してください。	次の内容を示す HTTPs 400 エラーを探します。
		• 無効または失効したクライアント証明書が提供された場合。
		• スイッチに構成されている CRL の有効期限が切れている場合。
		次に例を示します。
		`<html>`
		`<head><title>400 The SSL certificate error</title></head>`
		`<body bgcolor="white">`
		`<center><h1>400 Bad Request</h1></center>`
		`<center>The SSL certificate error</center>`
		`<hr><center>nginx/1.7.10</center>`
		`</body>`
		`</html>`

Step 3	エラーが発生した場合は、 **no nxapi client certificate authentication** 、それから **nxapi client certificate authentication** を発行して、トラストポイント、CA、CRL、または NX-OS 証明書機能に対する変更をリロードするように機能をフラップします。	証明書認証を無効にしてから、再度有効にします。

NX-API リクエスト要素

NX-API リクエスト要素は、XML フォーマットまたは JSON フォーマットでデバイスに送信されます。リクエストの HTTP ヘッダーは、リクエストのコンテンツ タイプを識別する必要があります。

次の表にリストされている NX-API 要素を使用して、CLI コマンドを指定します。

（注） ユーザーには、「configure terminal」コマンドを実行する権限が必要です。JSON-RPC が入力リクエスト フォーマットの場合、「configure terminal」コマンドは、常に、ペイロード内のコマンドが実行される前に実行されます。

表 *2 : XML* または *JSON* フォーマットの *NX-API* リクエスト要素

NX-API リクエスト要素	説明
version	NX-API バージョンを指定します。

NX-API リクエスト要素	説明
type	実行するコマンドのタイプを指定します。 次のタイプのコマンドがサポートされています。 • **cli_show** 構造化された出力が必要なCLI **show** コマンド。コマンドが XML 出力をサポートしていない場合は、エラーメッセージが返されます。 • **cli_show_ascii** ASCII出力が必要なCLI **show** コマンド。これは、ASCII出力を解析する既存のスクリプトと一致します。ユーザーは、最小限の変更で既存のスクリプトを使用できます。 • **cli_conf** CLI 構成コマンド • **bash** Bashコマンド。ほとんどの非対話型Bashコマンドは、NX-API でサポートされています。 （注） • 各コマンドは、現在のユーザーの権限でのみ実行可能です。 • メッセージタイプがASCIIの場合、出力でパイプ操作がサポートされます。出力がXML 形式の場合、パイプ操作はサポートされていません。 • 最大10の連続する **show** コマンドがサポートされています。**show** コマンドの数が 10 を超える場合、11番目以降のコマンドは無視されます。 • 対話型コマンドはサポートされていません。

NX-API リクエスト要素	説明
チャンク	一部の **show** コマンドは、大量の出力を返す場合があります。コマンド全体が完了する前に NX-API クライアントが出力の処理を開始するために、NX-API は **show** コマンドの出力チャンクをサポートしています。 次の設定を有効または無効にできます。 （注） 0 チャンク出力しません。 1 チャンク出力。 （注） • チャンクをサポートするのは **show** コマンドだけです。一連の **show** コマンドが入力されると、最初のコマンドだけがチャンクされて返されます。 • 出力メッセージ形式のオプションは、XML または JSON です。 • XML 出力メッセージ形式の場合< または> などの特殊文字は、有効な XML メッセージを形成するために変換されます（< は < に変換されます > は > に変換されます）。 XML SAX を使用して、チャンクされた出力を解析できます。 • 出力メッセージ形式が JSON の場合、チャンクが連結されて有効な JSON オブジェクトが作成されます。 （注）チャンクが有効になっている場合、現在サポートされている最大メッセージサイズは、チャンク出力の 200MB です。

<table>
<tr><th>NX-API リクエスト要素</th><th>説明</th></tr>
<tr><td>sid</td><td>セッション ID 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。メッセージの次のチャンクを取得するには、前の応答メッセージの sid と一致する sid を指定する必要があります。

NX-OS リリース 9.3(1) では、sid オプション <code>clear</code> が導入されています。sid を <code>clear</code> に設定して新しいチャンク リクエストが開始されると、現在のチャンク リクエストはすべて破棄または破棄されます。

応答コード <code>429</code> を受け取った場合：同時チャンク リクエストの最大数は <code>2</code> です。sid <code>clear</code> を使用して、現在のチャンク リクエストを破棄します。sid <code>clear</code> を使用した後、後続の応答コードは、残りのリクエストに対して通常どおり動作します。</td></tr>
<tr><td>input</td><td>入力は1つのコマンドまたは複数のコマンドです。ただし、異なるメッセージ タイプに属するコマンドを混在させてはなりません。たとえば、show コマンドは cli_show メッセージ タイプであり、cli_conf モードではサポートされません。

（注） bash を除き、複数のコマンドは「;」で区切ります。（; は、単一のブランク文字で囲む必要があります。）

エラーコード 500 でタイムアウトしないように、コマンドの前に <code>terminal dont-ask</code> を付加します。次に例を示します。

<code>terminal dont-ask ; cli_conf ; interface Eth4/1 ; no shut ; switchport</code>

bash の場合、複数のコマンドは「;」で区切ります。（; は単一のブランク文字で囲まれていません。）

以下は、複数のコマンドの例です。

（注）
<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>
</table>

<table>
<tr><th>NX-API リクエスト要素</th><th>説明</th></tr>
<tr><td>output_format</td><td>使用可能な出力メッセージ形式は次のとおりです。
（注）
<table>
<tr><td>xml</td><td>XML 形式を指定します。</td></tr>
<tr><td>json</td><td>JSON 形式で出力を指定します。</td></tr>
</table>
（注） Cisco NX-OS CLI は XML 出力をサポートしています。つまり、JSON 出力は XML から変換されます。変換はスイッチで処理されます。
計算のオーバーヘッドを管理するために、JSON 出力は出力の量によって決定されます。出力が 1 MB を超える場合、出力は XML 形式で返されます。出力がチャンクされている場合、XML 出力のみがサポートされます。
HTTP/HTTPS ヘッダーの content-type ヘッダーは、応答形式（XML または JSON）のタイプを示します。</td></tr>
</table>

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 *3 : 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。
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。

NX-API 応答要素	説明
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。

NX-API へのアクセスの制限

デバイスへの HTTP および HTTPS アクセスを制限するには、ACL と iptable の 2 つの方法があります。使用する方法は、`nxapi use-vrf<vrf-name>` CLI コマンドを使用して、NX-API 通信の VRF を構成していたかどうかに応じて決まります。

特定の VRF を使用するように NXAPI を構成していない場合にのみ、ACL を使用してデバイスへの HTTP または HTTPS アクセスを制限します。ACL の構成の詳細については、使用しているスイッチ ファミリの *Cisco Nexus* シリーズ *NX-OS* セキュリティ構成ガイドを参照してください。

ただし、NX-API 通信用に VRF を設定した場合、ACL は HTTP または HTTPS アクセスを制限しません。代わりに、iptable のルールを作成します。ルールの作成の詳細については、iptable の更新 （23 ページ）を参照してください。

iptable の更新

iptable を使用すると、VRF が NX-API 通信用に構成されている場合に、デバイスへの HTTP または HTTPS アクセスを制限できます。このセクションでは、既存の iptable に HTTP および HTTPS アクセスをブロックするルールを追加、確認、削除する方法を示します。

Step 1 HTTP アクセスをブロックするルールを作成するには、次の手順を実行します。

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

Step 2 HTTPS アクセスをブロックするルールを作成するには、次の手順を実行します。

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

Step 3 適用されたルールを確認するには、次の手順を実行します。

```
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 10.155.0.0/24 サブネットのポート 80 へのすべてのトラフィックをブロックするルールを作成して確認するには、次の手順を実行します。

```
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 以前に適用したルールを削除して確認するには、次の手順を実行します。

この例では、最初のルールを INPUT から削除します。

```
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
```

次のタスク

iptables のルールを bash シェルで変更した場合、リロード後は保持されません。ルールを永続的にするには、リロード間で Iptable を永続化する （24 ページ）を参照してください。

リロード間で Iptable を永続化する

iptable のルールを bash シェルで変更した場合、リロード後は保持されません。このセクションでは、リロード後も変更された iptable を永続化する方法について説明します。

始める前に

iptable を変更したとします。

Step 1 iptables_init.log という名前のファイルを /etc ディレクトリに作成します。

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

Step 2 iptable の変更を保存する /etc/sys/iptables ファイルを作成します。

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

Step 3 次の一連のコマンドを使用して、/etc/init.dディレクトリに「iptables_init」という起動スクリプトを作成します。

```
#!/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 起動スクリプトに適切な権限を設定します。

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

Step 5 chkconfig ユーティリティを使用して、「iptables_int」起動スクリプトを「オン」に設定します。

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

「iptables_init」起動スクリプトは、リロードを実行するたびに実行されます。これで iptable ルールを永続的にすることができました。

NX-API 応答コードの表

（注） 標準の HTTP エラー コードは、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) にあります。

表 *4 : NX-API* 応答コード

[NX-API 応答（NX-API Response）]	コード	メッセージ
成功	200	成功。
CUST_OUTPUT_PIPED	204	要求により、出力は別の場所にパイプされます。
BASH_CMD_ERR	400	Bash コマンド エラー
CHUNK_ALLOW_ONE_CMD_ERR	400	チャンクは、1 つのコマンドだけを受け入れます。
CLI_CLIENT_ERR	400	CLI の実行エラー
CLI_CMD_ERR	400	CLI コマンド エラーの入力。
EOC_NOT_ALLOWED_ERR	400	eoc 値は、リクエストのセッション ID として許可されていません。
IN_MSG_ERR	400	着信メッセージが無効です。
INVALID_REMOTE_IP_ERR	400	要求のリモートIPを取得できません。
MSG_VER_MISMATCH	400	メッセージ バージョンの不一致
NO_INPUT_CMD_ERR	400	入力コマンドがありません。

SID_NOT_ALLOWED_ERR	400	セッション ID として入力された文字が無効です。
PERM_DENY_ERR	401	権限が拒否されました。
CONF_NOT_ALLOW_SHOW_ERR	405	構成モードは **[表示（show）]** を許可しません。
SHOW_NOT_ALLOW_CONF_ERR	405	表示モードでは構成できません。
EXCEED_MAX_SHOW_ERR	413	連続する show コマンドの最大数を超えました。最大値は 10 です。
MSG_SIZE_LARGE_ERR	413	応答サイズが大きすぎます。
RESP_SIZE_LARGE_ERR	413	応答サイズが最大メッセージ サイズを超えたため、処理を停止しました。最大サイズは 200 MB です。
EXCEED_MAX_INFLIGHT_CHUNK_REQ_ERR	429	同時チャンク リクエストの最大数は超えています。最大は 2 です。
MAX_SESSIONS_ERR	429	最大セッション数に到達しました。新しいユーザー/クライアントの場合は、しばらくしてからもう一度お試しください。
OBJ_NOT_EXIST	432	要求したオブジェクトが存在しません。
BACKEND_ERR	500	バックエンド処理エラー。
CREATE_CHECKPOINT_ERR	500	チェックポイントの作成をするエラー。
DELETE_CHECKPOINT_ERR	500	チェックポイントの削除中にエラーが発生しました。
FILE_OPER_ERR	500	システム内部ファイル操作エラー。
LIBXML_NS_ERR	500	システムの内部 LIBXML NS エラー。これは要求フォーマットのエラーです。
LIBXML_PARSE_ERR	500	システムの内部 LIBXML 解析エラー。これは要求フォーマットのエラーです。
LIBXML_PATH_CTX_ERR	500	システムの内部 LIBXML パス コンテキスト エラー。これは要求フォーマットのエラーです。
MEM_ALLOC_ERR	500	システムの内部メモリ割り当てエラー。

ROLLBACK_ERR	500	ロールバックの実行中にエラーが発生しました。
SERVER_BUSY_ERR	500	サーバーがビジー状態のため、リクエストは拒否されました。
USER_NOT_FOUND_ERR	500	入力またはキャッシュからユーザーが見つかりません。
VOLATILE_FULL	500	揮発性メモリは一杯です。メモリ スペースを解放して、再試行してください。
XML_TO_JSON_CONVERT_ERR	500	XML から JSON への変換エラー。
BASH_CMD_NOT_SUPPORTED_ERR	501	Bash コマンドはサポートされていません。
CHUNK_ALLOW_XML_ONLY_ERR	501	チャンクは XML 出力のみを許可します。
CHUNK_ONLY_ALLOWED_IN_SHOW_ERR	501	応答のチャンクは、`show` コマンドでのみ許可されます。
CHUNK_TIMEOUT	501	チャンク応答の生成中にタイムアウトしました。
CLI_CMD_NOT_SUPPORTED_ERR	501	CLI コマンドはサポートされていません。
JSON_NOT_SUPPORTED_ERR	501	大量の出力の可能性があるため、JSONはサポートされていません。
MALFORMED_XML	501	不正な XML 出力。
MSG_TYPE_UNSUPPORTED_ERR	501	メッセージ タイプはサポートされていません
OUTPUT_REDIRECT_NOT_SUPPORTED_ERR	501	出力リダイレクトはサポートされていません。
PIPE_XML_NOT_ALLOWED_IN_INPUT	501	このコマンドへのパイプXMLは入力では許可されていません。
PIPE_NOT_ALLOWED_IN_INPUT	501	この入力タイプにはパイプを使用できません。
RESP_BIG_USE_CHUNK_ERR	501	応答が許容最大値を超えています。最大は 10 MB です。チャンクを有効にして XML または JSON 出力を使用します。
STRUCT_NOT_SUPPORTED_ERR	501	構造化出力はサポートされていません。
ERR_UNDEFINED	600	不明なエラー。

JSON および XML 構造化出力

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 出力の上限は 60 MB です。
- JSON フォーマット出力の標準規格ブロックを読みやすくした JSON Pretty もあります。JSON 出力の上限は 60 MB です。
- NX-OS リリース 9.3（1）で導入された JSON Native と JSON Pretty Native は、追加のコマンド解釈レイヤーをバイパスすることにより、JSON 出力をより高速かつ効率的に表示します。JSON Native および JSON Pretty Native は、出力のデータ型を保持します。出力用の文字列に変換する代わりに、整数を整数として表示します。

NX-OS CLI で、標準の NX-OS 出力を JSONまたは XML インタープリターに「パイプ接続」すると、これらのフォーマットへの変換が行われます。たとえば、**show ip access** コマンドを発行する際、論理パイプ（|）を続けて、その後に出力形式を指定できます。こうすると、NX-OS コマンドの出力が適切に構造化され、その形式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、JSON ネイティブ、JSON ネイティブ Pretty、および XML 出力をサポートしています。整合性チェッカー コマンドなど、一部のコマンドは、すべての形式をサポートしてはいません。整合性チェッカー コマンドは XML をサポートしていますが、JSON のバリアントはどれもサポートしていません。

（注） 検証エラーを回避するには、ファイル リダイレクトを使用して JSON 出力をファイルにリダイレクトし、そのファイル出力を使用します。

例:

```
Switch#show version | json > json_output ; run bash cat /bootflash/json_output
```

この機能の選択された例を以下に表示します。

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。コマンド出力では、JSON および JSON プリティ フォーマット、および JSON ネイティブおよび JSON プリティ ネイティブがサポートされています。

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 または XML 出力には、NX-API サンドボックスからもアクセスできます。

CLI の実行

```
switch-1-vxlan-1# show cdp neighbors  | json
{"TABLE_cdp_neighbor_brief_info": {"ROW_cdp_neighbor_brief_info": [{"ifindex": "
83886080", "device_id": "SW-SWITCH-1", "intf_id": "mgmt0", "ttl": "148"
, "capability": ["switch", "IGMP_cnd_filtering"], "platform_id": "cisco AA-C0000
S-29-L", "port_id": "GigabitEthernet1/0/24"}, {"ifindex": "436207616", "device
_id": "SWITCH-1-VXLAN-1(FOC1234A01B)", "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 出力の例

このセクションでは、XML および JSON 出力として表示される NX-OS コマンドの例について説明します。

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方法を示します。

```
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方法を示します。

```
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)#
```

この例は、スイッチの冗長性情報を JSON Pretty Native 形式で表示する方法を示しています。

```
switch-1# show system redundancy status | json-pretty native
{
        "rdn_mode_admin":       "HA",
        "rdn_mode_oper":        "None",
        "this_sup":     "(sup-1)",
        "this_sup_rdn_state":   "Active, SC not present",
        "this_sup_sup_state":   "Active",
        "this_sup_internal_state":      "Active with no standby",
        "other_sup":    "(sup-1)",
        "other_sup_rdn_state":  "Not present"
}
switch-1#
```

次の例は、スイッチの OSPF ルーティング パラメータを JSON ネイティブ形式で表示する方法を示しています。

```
switch-1# show ip ospf | json native
{"TABLE_ctx":{"ROW_ctx":[{"ptag":"Blah","instance_number":4,"cname":"default","
rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr
_grace_period":"PT60S","gr_state":"inactive","gr_last_status":"None","support_t
os0_only":"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false"
,"admin_dist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S"
,"spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_
time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"m
ax_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cn
t":0,"asopaque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_
nssa":0,"act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa
":0,"no_discard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"100","ins
tance_number":3,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":
"true","gr_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive"
,"gr_last_status":"None","support_tos0_only":"true","support_opaque_lsa":"true"
,"is_abr":"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_
time":"PT0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0
S","lsa_hold_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_
aging_pace":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"
asext_lsa_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"
area_normal":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal"
:0,"act_area_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_
rt_int":"false"},{"ptag":"111","instance_number":1,"cname":"default","rid":"0.0
.0.0","stateful_ha":"true","gr_ha":"true","gr_planned_only":"true","gr_grace_pe
riod":"PT60S","gr_state":"inactive","gr_last_status":"None","support_tos0_only"
:"true","support_opaque_lsa":"true","is_abr":"false","is_asbr":"false","admin_d
ist":110,"ref_bw":40000,"spf_start_time":"PT0S","spf_hold_time":"PT1S","spf_max
_time":"PT5S","lsa_start_time":"PT0S","lsa_hold_time":"PT5S","lsa_max_time":"PT
5S","min_lsa_arr_time":"PT1S","lsa_aging_pace":10,"spf_max_paths":8,"max_metric
_adver":"false","asext_lsa_cnt":0,"asext_lsa_crc":"0","asopaque_lsa_cnt":0,"aso
paque_lsa_crc":"0","area_total":0,"area_normal":0,"area_stub":0,"area_nssa":0,"
act_area_total":0,"act_area_normal":0,"act_area_stub":0,"act_area_nssa":0,"no_d
iscard_rt_ext":"false","no_discard_rt_int":"false"},{"ptag":"112","instance_num
ber":2,"cname":"default","rid":"0.0.0.0","stateful_ha":"true","gr_ha":"true","g
r_planned_only":"true","gr_grace_period":"PT60S","gr_state":"inactive","gr_last
_status":"None","support_tos0_only":"true","support_opaque_lsa":"true","is_abr"
:"false","is_asbr":"false","admin_dist":110,"ref_bw":40000,"spf_start_time":"PT
0S","spf_hold_time":"PT1S","spf_max_time":"PT5S","lsa_start_time":"PT0S","lsa_h
old_time":"PT5S","lsa_max_time":"PT5S","min_lsa_arr_time":"PT1S","lsa_aging_pac
e":10,"spf_max_paths":8,"max_metric_adver":"false","asext_lsa_cnt":0,"asext_lsa
_crc":"0","asopaque_lsa_cnt":0,"asopaque_lsa_crc":"0","area_total":0,"area_norm
al":0,"area_stub":0,"area_nssa":0,"act_area_total":0,"act_area_normal":0,"act_a
rea_stub":0,"act_area_nssa":0,"no_discard_rt_ext":"false","no_discard_rt_int":"
false"}]}}
switch-1#
```

次の例は、OSPF ルーティング パラメータを JSON Pretty Native 形式で表示する方法を示しています。

```
switch-1# show ip ospf | json-pretty native
{
   "TABLE_ctx": {
          "ROW_ctx": [{
                        "ptag": "Blah",
                        "instance_number":      4,
                        "cname":        "default",
                        "rid":  "0.0.0.0",
                        "stateful_ha":  "true",
                        "gr_ha":        "true",
                        "gr_planned_only":      "true",
                        "gr_grace_period":      "PT60S",
                        "gr_state":     "inactive",
                        "gr_last_status":       "None",
                        "support_tos0_only":    "true",
                        "support_opaque_lsa":   "true",
                        "is_abr":       "false",
                        "is_asbr":      "false",
                        "admin_dist":   110,
                        "ref_bw":       40000,
                        "spf_start_time":       "PT0S",
                        "spf_hold_time":        "PT1S",
                        "spf_max_time": "PT5S",
                        "lsa_start_time":       "PT0S",
                        "lsa_hold_time":        "PT5S",
                        "lsa_max_time": "PT5S",
                        "min_lsa_arr_time":     "PT1S",
                        "lsa_aging_pace":       10,
                        "spf_max_paths":        8,
                        "max_metric_adver":     "false",
                        "asext_lsa_cnt":        0,
                        "asext_lsa_crc":        "0",
                        "asopaque_lsa_cnt":     0,
                        "asopaque_lsa_crc":     "0",
                        "area_total":   0,
                        "area_normal":  0,
                        "area_stub":    0,
                        "area_nssa":    0,
                        "act_area_total":       0,
                        "act_area_normal":      0,
                        "act_area_stub":        0,
                        "act_area_nssa":        0,
                        "no_discard_rt_ext":    "false",
                        "no_discard_rt_int":    "false"
                      }, {
                        "ptag": "100",
                        "instance_number":      3,
                        "cname":        "default",
                        "rid":  "0.0.0.0",
                        "stateful_ha":  "true",
                        "gr_ha":        "true",
                        "gr_planned_only":      "true",
                        "gr_grace_period":      "PT60S",
                        "gr_state":     "inactive",

                        ... content deleted for brevity ...

                        "max_metric_adver":     "false",
                        "asext_lsa_cnt":        0,
                        "asext_lsa_crc":        "0",
                        "asopaque_lsa_cnt":     0,
                        "asopaque_lsa_crc":     "0",
                        "area_total":   0,
                        "area_normal":  0,
```

```
                        "area_stub":    0,
                        "area_nssa":    0,
                        "act_area_total":       0,
                        "act_area_normal":      0,
                        "act_area_stub":        0,
                        "act_area_nssa":        0,
                        "no_discard_rt_ext":    "false",
                        "no_discard_rt_int":    "false"
                }]
        }
}
switch-1#
```

次の例は、JSON ネイティブ形式で IP ルートテーブルを表示する方法を示しています。

```
switch-1(config)# show ip route summary | json native
{"TABLE_vrf":{"ROW_vrf":[{"vrf-name-out":"default","TABLE_addrf":{"ROW_addrf":[{"addrf":"ipv4","TABLE_summary":{"ROW_summary":[{"routes":3,"p
aths":3,"TABLE_unicast":{"ROW_unicast":[{"clientnameuni":"broadcast","best-paths":3}]},"TABLE_route_count":{"ROW_route_count":[{"mask_len":8,
"count":1},{"mask_len":32,"count":2}]}}]}}]}}]}}
switch-1(config)#
```

JSON ネイティブ（および JSON プリティ ネイティブ）では、整数が真の整数として表されることに注意してください。たとえば、「mask len:」は実際の値 32 として表示されます。

次の例は、JSON プリティ ネイティブ形式で IP ルートテーブルを表示する方法を示しています。

```
switch-1(config)# show ip route summary | json-pretty native
{
  "TABLE_vrf": {
    "ROW_vrf": [{
        "vrf-name-out": "default",
        "TABLE_addrf": {
               "ROW_addrf": [{
                    "addrf": "ipv4",
                    "TABLE_summary": {
                           "ROW_summary": [{
                                "routes": 3,
                                "paths": 3,
                              "TABLE_unicast": {
                                    "ROW_unicast": [{
                                          "clientnameuni": "broadcast",
                                          "best-paths": 3
                                                    }]
                                                },
                                                 "TABLE_route_count": {
                                                       "ROW_route_count":[{
                                                             "mask_len": 8,
                                                             "count": 1
                                                                   }, {
                                                             "mask_len": 32,
                                                             "count": 2
                                                       }]
                                                }
                                          }]
                                    }
                           }]
                    }
               }]
        }
}
switch-1(config)#
```

サンプル NX-API スクリプト

ユーザーはNX-APIでスクリプトを使用する方法を示すサンプル スクリプトにアクセスできます。サンプル スクリプトにアクセスするには、次のリンクをクリックして、必要なソフトウェア リリースに対応するディレクトリを選択します：Cisco Nexus 9000 NX-OS NX-API

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。