NETCONF エージェント

この章は次のトピックで構成されています。

NETCONF エージェントについて

NETCONF（Network Configuration Protocol、ネットワーク構成プロトコル）は、RFC 6241 によって定義されているネットワーク管理プロトコルです。Cisco NX-OS は、クライアント側のインターフェイスである NETCONF エージェントを提供しており、XML でエンコードされた YANG モデルの形式で、クライアントの要求とサーバーの応答のため、SSH 上のセキュアな転送を提供します。

NETCONF は、構成データストアと、これらのデータストアでの操作とクエリを可能にする一連の作成、読み取り、更新、および削除（CRUD）操作を定義しています。NX-OS では、実行、起動、候補の 3 つのデータストアがサポートされています。サポートされている操作の簡単な説明を次に示します。

表 *1*: サポートされる操作

動作	説明
get	実行構成と動作状態を取得します。
get-config	指定されたデータストアから構成を取得します。

動作	説明
edit-config	指定されたターゲット データストアに、指定された構成をロードします
close-session	セッションの適切な終了を要求します。
kill-session	セッションを強制終了します。
copy-config	別のデータストアの内容でデータストアを作成するか、置き換えます。
ロック	データストアをロックします。
unlock	データストアのロックを解除します。
検証	指定された構成の内容を検証します。
commit	候補構成を新しい現行の実行構成としてコミットします。
cancel-commit	進行中の要確認コミットをキャンセルします。
discard-changes	候補構成を現行の実行構成に戻します。

NETCONF に関する注意事項と制限事項

NETCONF エージェントには、次の注意事項と制限事項があります。

- Cisco NX-OS は、NETCONF 通知で Cisco デバイス YANG モデルと OpenConfig モデルの両方をサポートします。
- デバイス YANG モデルはエフェメラルデータを定義します。これらは「// Ephemeral data」というコメントでマークされます。これらの非永続的な大容量データは、モデルの残りの部分とは異なる方法で処理されます。これらは、<get> クエリの <filter> パラメータが、コメントでマークされた特定の要素を具体的に指している場合にのみ返されます。使用方法の詳細については、エフェメラルデータ サポートのドキュメントを参照してください。
- Cisco NX-OS リリース 9.3(3) 以降、NETCONF は RFC 6241 に準拠していますが、次の例外があります。
  - 兄弟コンテンツマッチ ノードは、「AND」式ではなく「OR」式で論理的に結合されます。（セクション 6.2.5）
  - 候補データストアを編集した後は、同じプロパティの実行構成を編集しないでください。
- 1 つの Get 要求でサポートされるオブジェクトの数は 250,000 です。次のエラーが表示された場合は、要求されたデータが 250,000 を超えていることを意味します。このエラーを

回避するには、データをさらに絞り込んでクエリするためのフィルタを使用して要求を送信します。

```
too many objects(459134 > 250000) to query the entire device model.
```

- NETCONF は、RFC 6536 で指定されている拡張ロールベース アクセス コントロール（RBAC）をサポートしていません。「network-admin」ロールを持つユーザーのみがNETCONF エージェントへのアクセスを許可されます。
- `<edit-config>` の「置換」操作は、影響を受けるシステム コンポーネントによって実装されている実行時デフォルト値と動作が原因で、機能しない場合があります。したがって、NX-API 開発者サンドボックスの代わりに、`<get-config>` クエリによって取得した構成上で、置換を行うための構成を基礎とする方が適切です。
- Cisco NX-OS NETCONF サーバは、最大 5 つのサブスクリプション（クライアント セッションごとに 1 つのサブスクリプション）をサポートします。
- RFC 5277 によれば、自律通知は、イベント ソースの NETCONF、SYSLOG、および SNMP ストリームをサポートします。このリリースでは、Cisco NX-OS は NETCONF ストリームのみをサポートします。
- Cisco NX-OS は、サブスクリプションの [再生（Replay）] オプションをサポートしていません。[開始時刻（Start Time）] オプションと [終了時刻（Stop Time）] オプションは再生の一部であるため、サポートされていません。
- ストリーム サブスクリプションとフィルタリングでは、サブツリー フィルタリングのみがサポートされます。XPath フィルタリングはサポートされていません。
- Cisco NX-OS NETCONF エージェントが高負荷で動作している場合、一部のイベント通知がドロップされる可能性があります。
-
- Cisco NX-OS は、Cisco デバイス YANG モデルと OpenConfig モデルの両方をサポートします。NETCONF 通知で OpenConfig モデルをサポートするのは、Cisco NX-OS 9.3(5) リリース以降です。
- 10.2(1)F リリース以降では、操作チェックポイント、ロールバック、インストール、CA 証明書のインポート、モジュールのリロード、個々のモジュールのリロード、およびファイルのコピーがサポートされています。
- L2 MAC リーフプロパティ値を入力として、openconfig-acl NETCONF GET 操作を実行する場合は、MAC アドレスの文字を大文字形式（AA:AA: AA:AA:AA:AA）で入力することをお勧めします。たとえば、source-mac: 0A:0B:0C:0D:0E:0F です。

NETCONF エージェントの構成

Cisco NX-OS 9.3(5) 以降の SSH を介した NETCONF エージェントの設定

この手順では、SSHを介してNETCONFエージェントを有効にして構成する方法について説明します。

（注） この手順は、Cisco NX-OS リリース 9.3(5) 以降で使用します。

始める前に

NETCONF を使用してスイッチと通信する前に、NETCONF エージェントを有効にする必要があります。NETCONF エージェントを有効または無効にするには、**[no] feature netconf** コマンドを入力します。

手順の概要

1. **configure terminal**
2. **feature netconf**
3. （任意） **netconf idle-timeout** *it-num*
4. （任意） **netconf sessions** *num-sessions*

手順の詳細

	コマンドまたはアクション	目的
ステップ 1	**configure terminal** 例： `switch# configure terminal`	グローバル コンフィギュレーション モードを開始します。
ステップ 2	**feature netconf** 例： `switch(config)# feature netconf`	NETCONF サービスを有効にします。
ステップ 3	（任意） **netconf idle-timeout** *it-num* 例： `switch(config)# netconf idle-timeout 5`	（オプション）アイドル状態のクライアント セッションが切断されるまでのタイムアウトを分単位で指定します。*it-num* の範囲は 0 ～ 1440 分です。デフォルトのタイムアウトは 5 分です。値を 0 に設定するとタイムアウトが無効になります。

	コマンドまたはアクション	目的
ステップ 4	（任意） **netconf sessions** *num-sessions* 例： `switch(config)# netconf sessions 5`	同時クライアント セッションの最大数を指定します。*num-sessions* の範囲は 1 ～ 10 です。デフォルト値は 5 セッションです。

Cisco NX-OS 9.3(4) 以前の NETCONF エージェントの構成

（注） Cisco NX-OS リリース9.3(4) 以前の場合は、次の手順に従ってください。

NETCONFエージェントは、構成ファイル（/etc/mtx.conf）の [netconf] セクションで、次のオプションの構成パラメータをサポートします。

パラメータ	説明
idle_timeout	（オプション）アイドル状態のクライアント セッションが切断されるまでのタイムアウトを分単位で指定します。 デフォルト値は 5 分です。 値を 0 に設定するとタイムアウトが無効になります。
limit	（オプション）同時クライアント セッションの最大数を指定します。 デフォルト値は 5 セッションです。 指定できる範囲は、1 ～ 10 です。

次に、構成ファイルの [netconf] セクションの例を示します。

```
[netconf]
mtxadapter=/opt/mtx/lib/libmtxadapternetconf.1.0.1.so
idle_timeout=10
limit=1
```

変更した構成ファイルを有効にするには、CLI コマンド **[no] feature netconf** を使用して NETCONF エージェントを無効にしてから再度有効にして、再起動する必要があります。

NETCONF セッションの確立

NETCONFは、クライアントとサーバー間の永続的な接続を必要とする接続指向のプロトコルです。スイッチ上の NETCONF エージェントは、管理ポート IP アドレスのポート 830 でリッ

スンします。クライアントは、SSH を介して NETCONF サブシステムとの接続を確立できます。クライアントが NETCONF エージェントとのセッションを確立すると、サーバーは <hello> メッセージをクライアントに送信します。同様に、クライアントは <hello> メッセージをサーバーに送信します。<hello> メッセージは、接続が開くと同時に交換されます。各 <hello> メッセージには、送信側ピアのプロトコルバージョンと機能のリストが含まれています。これらのメッセージは、プロトコルの互換性と機能を判断するために使用されます。両方の NETCONF ピアは、相手の <hello> メッセージで、共通のプロトコルバージョンがアドバタイズされたかどうかを確認する必要があります。また、サーバーの <hello> メッセージには <session-id> を含める必要がありますが、クライアントの <hello> メッセージには含めることができません。

次に、**ssh** コマンドを使用したセッション確立の例を示します。最初の <hello> メッセージがサーバーから受信され、2 番目のメッセージがクライアントから送信されます。サーバーの <hello> メッセージには、プロトコル バージョンである「urn:ietf:params:netconf:base:1.1」と、Cisco NX-OS リリース 9.3(4) でサポートされている NETCONF の基本機能が含められます。また、サーバーの <hello> メッセージには、サポートされているデータ モデルが含められます。これらは、現在の Cisco NX-OS リリースでサポートされているモデルとは一致しない場合があります。

（注） サーバーの <hello> メッセージには <session-id> が含められますが、クライアントのメッセージには含められません。

```
client-host % ssh admin@172.19.193.166 -p 830 -s netconf
User Access Verification
Password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
        <capability>urn:ietf:params:netconf:base:1.0</capability>
        <capability>urn:ietf:params:netconf:base:1.1</capability>
       <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>

       <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>

        <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
       <capability>urn:ietf:params:netconf:capability:confirmed-commit:1.1</capability>

        <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
        <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>

<capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=report-all</capability>


<capability>http://cisco.com/ns/yang/cisco-nx-os-device?revision=2020-04-20&module=Cisco-NX-OS-device</capability>


<capability>http://openconfig.net/yang/acl?revision=2019-11-27&module=openconfig-acl&deviations=cisco-nx-openconfig-acl-deviations</capability>


<capability>http://openconfig.net/yang/bfd?revision=2019-10-25&module=openconfig-bfd&deviations=cisco-nx-openconfig-bfd-deviations</capability>

        ...
    </capabilities>
    <session-id>1286775422</session-id>
```

```
</hello>
]]>]]><hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
  </capabilities>
</hello>
]]>]]>
```

RFC 6242（SSH 上の NETCONF プロトコルの使用）からわかるとおり、メッセージ フレーミングが複雑であるため、**ssh** コマンドで NETCONF を使用することは便利ではなく、エラーが発生しやすくなります。上記の例では、説明のみを目的として **ssh** コマンドを使用しています。NETCONF 用にはさまざまなクライアントが作成されており、**ssh** コマンドを使用するよりも推奨されています。ncclient はそのような例の 1 つであり、「使用例」のセクションで使用方法を説明します。

NETCONF は、セッションを終了するために、<close-session> および <kill-session> という 2 つの操作をサポートしています。サーバーは、<close-session> 要求を受信すると、セッションに関連付けられているロックとリソースを解放し、クライアントとの接続を閉じて、セッションを適切な方法で終了します。<close-session> 要求と応答が成功した例を次に示します。

```
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <close-session/>
</rpc>

<rpc-reply message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

<kill-session> 要求は別のセッションを強制的に終了するもので、要求メッセージ内で <session-id> を指定する必要があります。サーバーは、<kill-session> 要求を受信すると、現在の操作を終了し、指定されたセッション ID に関連したロックとリソースを解放し、接続を閉じます。<kill-session> 要求と応答が成功した例を次に示します。

```
<rpc message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <kill-session>
    <session-id>296324181</session-id>
  </kill-session>
</rpc>

<rpc-reply message-id="2" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

<close-session> および <kill-session> 要求とは別に、クライアントが一定時間要求を送信しなかった場合、セッションは自動的に終了します。デフォルトは 5 分です。アイドル タイムアウトの構成については、「NETCONF エージェントの構成」を参照してください。

NETCONF の読み取りおよび書き込み構成

このセクションでは、データストアの操作とクエリのためにサポートされている基本プロトコル操作について説明します。クライアントは、NETCONF エージェントとのセッションを確立

した後、これらの操作のための RPC メッセージを送信できます。ここでは基本的な使用方法について説明します。詳細については RFC 6242 を参照してください。

<get-config>

この操作により、指定したデータストアの構成データを取得します。サポートされるパラメータは <source> と <filter> です。<source> は、（現在アクティブな構成を保持している）<running/>のような、クエリ対象のデータストアを指定します。<filter>は、指定されたデータストアのうちどの部分を取得するかを指定します。

次に示すのは、<get-config> 要求と応答メッセージの例です。

- <System> サブツリー全体を取得します：

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
        <source>
            <running/>
        </source>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"/>
        </filter>
    </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        ...
        </System>
    </data>
</rpc-reply>
```

- 特定のリスト項目を取得します：

```
<rpc message-id="102" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get-config>
        <source>
            <running/>
        </source>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list>
                                <name>default</name>
                            </Dom-list>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </filter>
    </get-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="102">
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <bgp-items>
                <inst-items>
```

```
                    <dom-items>
                        <Dom-list>
                            <name>default</name>
                            ...
                            <rtctrl-items>
                                <enforceFirstAs>enabled</enforceFirstAs>
                                <fibAccelerate>disabled</fibAccelerate>
                                <logNeighborChanges>enabled</logNeighborChanges>
                                <supprRt>enabled</supprRt>
                            </rtctrl-items>
                            <rtrId>1.2.3.4</rtrId>
                        </Dom-list>
                    </dom-items>
                </inst-items>
            </bgp-items>
        </System>
    </data>
</rpc-reply>
```

<edit-config>

この操作は、指定された構成をターゲット データストアに書き込みます。<target> パラメータは、<running/>または<candidate/> など、編集するデータストアを指定します。候補データストアは、変更がコミットされるまで、実行中のデータストアに影響を与えることなく操作できます。詳細については、<commit> のセクションを参照してください。<config> パラメータは、ターゲットデータストアに書き込まれるモデル化されたデータを指定します。モデルは「xmlns」属性で指定されます。「operation」属性は、<config> サブツリーの任意の数の要素に含めることができます。要素の操作は、新しい「operation」属性によってオーバーライドされるまで、その子孫要素に継承されます。サポートされている操作は、「merge」、「replace」、「create」、「delete」、および「remove」です。「remove」操作は、設定データが存在しない場合にエラーが返されないという点で「delete」とは異なります。「operation」属性が指定されていない場合は、「merge」操作がデフォルトと見なされます。デフォルトの動作は、オプションの <default-operation> パラメータ（「merge」、「replace」、または「none」があります）によりオーバーライドできます。

次に示すのは、<edit-config> 要求と応答メッセージの例です。

- MTU が 9216 である「po5」という名前のポートチャネルを、実行構成の説明に基づいて作成します。

```
<rpc message-id="103" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="create">
                            <id>po5</id>
                            <mtu>9216</mtu>
                            <descr>port-channel 5</descr>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
```

```
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="103">
    <ok/>
</rpc-reply>
```

- ポートチャネルのすべての構成を新しい構成で置き換えます。

```
<rpc message-id="104" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
               <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="replace">
                            <id>po5</id>
                            <mtu>1500</mtu>
                            <adminSt>down</adminSt>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="104">
    <ok/>
</rpc-reply>
```

- ポートチャネルを削除します：

```
<rpc message-id="105" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
        <target>
            <running/>
        </target>
        <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
           <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <intf-items>
                    <aggr-items>
                        <AggrIf-list xc:operation="delete">
                            <id>po5</id>
                        </AggrIf-list>
                    </aggr-items>
                </intf-items>
            </System>
        </config>
    </edit-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="105">
    <ok/>
</rpc-reply>
```

<copy-config>

この操作は、ターゲットの構成データストアを、ソース構成データストア全体のコンテンツによって置き換えます。ソース データストアとターゲット データストアのパラメータはそれぞれ <source> と <target> です。

次に示すのは、<copy-config> 要求と応答メッセージの例です。

- 実行構成をスタートアップ構成にコピーします。

```
<rpc message-id="106" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
        <target>
            <startup/>
        </target>
        <source>
            <running/>
        </source>
    </copy-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="106">
    <ok/>
</rpc-reply>
```

- 実行構成を候補構成にコピーします。

```
<rpc message-id="107" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
        <target>
            <candidate/>
        </target>
        <source>
            <running/>
        </source>
    </copy-config>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="107">
    <ok/>
</rpc-reply>
```

<lock>

<lock>操作を使用すると、クライアントは設定データストアをロックし、他のクライアントがデータストアをロックまたは変更するのを防ぐことができます。クライアントが保持しているロックは、<unlock>操作を実行するか、セッションが終了すると、解除されます。<target>パラメータは、ロックするデータストアを指定します。

次に示すのは、<lock> 要求と応答メッセージの例です。

- ロックの取得に成功した場合：

```
<rpc message-id="108" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lock>
        <target>
            <running/>
        </target>
    </lock>
```

```
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="108">
    <ok/>
</rpc-reply>
```

- 別のセッションですでに使用されていたため、ロックの取得に失敗した場合：

```
<rpc message-id="109" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lock>
        <target>
            <candidate/>
        </target>
    </lock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="109">
    <rpc-error>
        <error-type>protocol</error-type>
        <error-tag>lock-denied</error-tag>
        <error-severity>error</error-severity>
       <error-message xml:lang="en">Lock failed, lock is already held</error-message>

        <error-info>
            <session-id>1553704357</session-id>
        </error-info>
    </rpc-error>
</rpc-reply>
```

<unlock>

<unlock> 操作は、以前に <lock> 操作によって取得した構成のロックを解除します。<lock> 操作を発行したのと同じセッションでのみ、<unlock> 操作を使用できます。<target> パラメータは、ロック解除するデータストアを指定します。

次に示すのは、<unlock> 要求と応答メッセージの例です。

- ロック解除

```
<rpc message-id="110" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <unlock>
        <target>
            <candidate/>
        </target>
    </unlock>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="110">
    <ok/>
</rpc-reply>
```

<get>

<get> 操作は、実行中の構成とデバイスの状態情報を取得します。サポートされているパラメータは <filter> です。<filter> パラメータは、実行構成の動作状態データのうち、どの部分を取得するかを指定します。

次に示すのは、<get> 要求と応答メッセージの例です。

- リスト項目の実行構成と動作状態データを取得します。

```
<rpc message-id="111" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter>
            <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
                        <dom-items>
                            <Dom-list>
                                <name>default</name>
                            </Dom-list>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </filter>
    </get>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="111">
    <data>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <bgp-items>
                <inst-items>
                    <dom-items>
                        <Dom-list>
                            <name>default</name>
                            <always>disabled</always>
                            <bestPathIntvl>300</bestPathIntvl>
                            <clusterId>120</clusterId>

<firstPeerUpTs>2020-04-20T16:19:03.784+00:00</firstPeerUpTs>
                            <holdIntvl>180</holdIntvl>
                            <id>1</id>
                            <kaIntvl>60</kaIntvl>
                            <mode>fabric</mode>
                            <numEstPeers>0</numEstPeers>
                            <numPeers>0</numPeers>
                            <numPeersPending>0</numPeersPending>
                            <operRtrId>1.2.3.4</operRtrId>
                            <operSt>up</operSt>
                            <pfxPeerTimeout>90</pfxPeerTimeout>
                            <pfxPeerWaitTime>90</pfxPeerWaitTime>
                            <reConnIntvl>60</reConnIntvl>
                            <rtrId>1.2.3.4</rtrId>
                            <vnid>0</vnid>
                            ...
                        </Dom-list>
                    </dom-items>
                </inst-items>
            </bgp-items>
        </System>
    </data>
</rpc-reply>
```

<validate>

この操作は、候補データストアの構成内容を検証します。これは、実行データストアにコミットする前に、候補データストアで行われた構成変更を検証するのに役立ちます。<source> パラメータは <candidate/> をサポートします。

次に示すのは、<validate> 要求と応答メッセージの例です。

- 候補データストアの内容を検証します：

```
<rpc message-id="112" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <validate>
        <source>
            <candidate/>
        </source>
    </validate>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="112">
    <ok/>
</rpc-reply>
```

<commit>

候補構成を実行構成にコミットします。パラメータを付けずに操作すると、最終的なものと見なされ、元に戻すことはできません。<commit> を <confirmed/> パラメータを付けて発行すると、要確認コミットであると見なされます。別の、<confirmed/> パラメータを付けない<commit>操作を続けた場合にのみ、コミットはファイナライズされます。このようにして、確認した上でコミットできます。要確認コミットでは、<confirm-timeout> と <persist> という 2 つのパラメータを使用できます。<confirm-timeout> は、要確認コミットが元に戻されるまでの秒数です。この時間が経過すると、実行構成は要確認コミットが発行される前の状態に復元されます。<confirm-timeout> が指定されていない場合、デフォルトのタイムアウトは 600 秒です。セッションが終了すると、要確認コミットは元に戻ります。<persist> パラメータを使用すると、セッションが終了しても要確認コミットが保持されます。<persist> パラメータの値は、任意のセッションからの要確認コミットを識別するために使用されます。そして後続の要確認コミットまたはこれから確認するコミットの <persist-id> の値として使用する必要があります。

次に示すのは、<commit> 要求と応答メッセージの例です。

- 候補データストアの内容をコミットします:

```
<rpc message-id="113" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="113">
    <ok/>
</rpc-reply>
```

- タイムアウトが経過した要確認コミット：

```
<rpc message-id="114" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <confirmed/>
```

```
        <confirm-timeout>120</confirm-timeout>
    </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="114">
    <ok/>
</rpc-reply>
```

- 永続的な要確認コミットを開始し、その後、永続的な要確認コミットを実際に確認します：

```
<rpc message-id="115" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <confirmed/>
        <persist>ID1234</persist>
    </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="115">
    <ok/>
</rpc-reply>

<!-- confirm the persistent confirmed-commit, from the same session or another session
 -->
<rpc message-id="116" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <commit>
        <persist-id>ID1234</persist-id>
    </commit>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="116">
    <ok/>
</rpc-reply>
```

<cancel-commit>

この操作は、進行中の要確認コミットをキャンセルします。別のセッションからの要確認コミットをキャンセルする必要がある場合、要確認コミットの <persist> パラメータで指定したのと同じ値を指定して、<persist-id> パラメータを使用する必要があります。

- 同じセッションの要確認コミットをキャンセルします：

```
<rpc message-id="117" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <cancel-commit/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="117">
    <ok/>
</rpc-reply>
```

<discard-changes>

この操作は、実行構成の内容にリセットすることによって、候補構成に加えられたコミットされていない変更を破棄します。パラメータは必要ありません。

次に示すのは、<discard-changes> 要求と応答メッセージの例です。

- 候補データストアに加えられた変更を破棄します。

```
<rpc message-id="118" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <discard-changes/>
</rpc>

<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="118">
    <ok/>
</rpc-reply>
```

NETCONF の実行

NETCONF でのモデル駆動型操作について

表 *2 : NETCONF* でのモデル駆動型操作について

オペレーション	NETCONF RPC	CLI
チェックポイント	checkpoint	checkpoint <name> checkpoint <file>
ロールバック	ロールバック	rollback running-config checkpoint <name> rollback running-config checkpoint <file>
インストールするもの	install_all_nxos install_add install_activate install_deactivate install_commit install_remove	install all nxos <image> install {add | activate | deactivate | commit | remove} <rpm>
暗号化証明書のインポート	import_ca_certificate	crypto ca import <trustpoint> pkcs12 <file> <passphrase>
スイッチのリロードまたはモジュールのリロード	reload	reload [timer <seconds>] reload module <module number>
ファイルのコピー	copy	copy <source> <destination>

モデル駆動型操作の例

モデル駆動型操作の例

ファイル名オプションを使用したチェックポイントの作成：

```
RPC:
<rpc message-id="checkpoint-3" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <file>bootflash:my_checkpoint2</file>
  </checkpoint>
</rpc>
```

チェックポイント名、説明を使用したチェックポイントの作成：

```
RPC:
<rpc message-id="checkpoint-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <action>create</action>
      <name>my_checkpoint1</name>
      <description>test checkpoint one</description>
  </checkpoint>
</rpc>
```

チェックポイント名を使用したチェックポイントの削除：

```
RPC:
<rpc message-id="delatecheckpoint-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <checkpoint xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
      <action>delete</action>
      <name>my_checkpoint1</name>
  </checkpoint>
</rpc>
```

ロールバック：

（注） 次のオプションタグは、アトミック、最初のエラーで停止、ベストエフォートとして使用できます。

```
<rpc message-id="rollback-cfg-option1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<rollback xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <name>my_checkpoint1</name>
    <option>atomic</option>
</rollback>
</rpc>
```

ファイル オプションを使用したロールバック

```
<rpc message-id="rollback-cfg1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><
<rollback xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
     <file>bootflash:my_checkpoint2</file>
</rollback>
</rpc>
```

ファイルのコピー

リモート サーバーからスイッチ ストレージに任意のファイルをコピーします（例：ブートフラッシュ）。

Kerry tftp の場合：プロトコルがファイル転送をサポートします。

```
<rpc message-id="copy-file-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<copy xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <source>tftp://172.27.xxx.xxx//<file_location?/tls1-server.pfx</source>
    <destination>bootflash:</destination>
      <vrf>management</vrf>
```

```
</copy>
</rpc>
```

CA 証明書のインポート

前提条件：スイッチで my_truspoint がすでに作成されている必要があります。

```
<rpc message-id="import_ca_certificate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<import_ca_certificate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <trustpoint>my_trustpoint</trustpoint>
    <pkcs12>tls1-server.pfx</pkcs12>
    <passphrase>xxxxxx</passphrase>
</import_ca_certificate>
</rpc>
```

RPM パッケージ インストール EXEC RPC コマンド.

Install <add>

```
<rpc message-id="install-add-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<install_add xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
 <add>rpm_packagenamehere_from_bootflash</add>
</install_add>
</rpc>
```

Install <activate>

```
<rpc message-id="install-activate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<install_activate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <activate> rpm_packagenamehere_from_bootflash</activate>
</install_activate>
</rpc>
```

Install <deactivate>

```
<rpc message-id="install-deactivate-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

  <install_deactivate xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <deactivate>rpm_packagenamehere_from_bootflash </deactivate>
 </install_deactivate>
</rpc>
```

Install <remove>

```
<rpc message-id="rpc-install_remove-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<install_remove xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
<remove>rpm_packagenamehere_from_bootflash </remove>
</install_remove>
</rpc>
```

すべての nx-os イメージのインストール

```
<rpc message-id="rpc-install_all_nxos-1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<install_all_nxos xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <nxos>nxos.image.bin.upg</nxos>
</install_all_nxos>
</rpc>
```

モジュール番号のリロード

```
<rpc message-id="reload-module-pyld1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<reload xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
    <module>29</module>
</reload>
</rpc>
```

再読み込み（Reload）

（注） クライアントが次の RPC を要求または送信すると、exec コマンドはスイッチのリロードを実行します。それ以上、Netconf クライアントは <ok> 応答を受信しません。

```
<rpc message-id="563" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<reload xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"/>
</rpc>
```

NETCONF 通知

NETCONF 通知について

NETCONF 通知は、NETCONF クライアントがシステムイベントにサブスクライブして、NETCONF エージェントからこれらのイベントに関する通知を受信できるようにするメカニズムです。これらの機能は、RFC 5277 で定義されています。Cisco NX-OS リリース 9.3(1) 以降、RFC 5277 で説明されているように、NETCONF 通知のサポートが開始されました。これは、NETCONF hello メッセージでアドバタイズされるオプションの機能です。

NETCONF クライアントは、Deviceyang または OpenConfig モデルを使用して、通知にサブスクライブできます。NETCONF 通知での OpenConfig モデルのサポートは、Cisco NX-OS リリース 9.3(5) から開始されました。

このサポートにより、NETCONF クライアントは次のことが行えます。

- イベント通知へのサブスクライブ

  各サブスクリプションは、NETCONF クライアントからのセッションを介した 1 回限りの要求です。Cisco NX-OS NETCONF エージェントが応答し、NETCONF クライアントによってセッションが明示的に閉じられるまで、サブスクリプションはアクティブです。サブスクリプションは、スイッチの再起動やスイッチの NETCONF 機能の無効化などの管理アクションによって閉じられることもあります。サブスクリプションは、基盤となる NETCONF セッションがアクティブである限りアクティブです。これらの登録済みフィルタに対して生成されたイベントは、通知としてクライアントに送信されます。クライアントは、システムイベントの通知にサブスクライブできます。たとえば、一部だけ挙げてみても、ポート状態の変更、ファン速度の変更、プロセスメモリの変更などがあります。また、有効になっている新機能などの構成イベントもあります。

- 障害イベント通知を受信します。

  イベント通知は、スイッチの設定イベントまたは動作イベントに関する情報を含む、形式の整った XML ドキュメントです。NETCONF クライアントは、サブスクリプション要求でフィルタリング基準を送信して、すべてのイベントではなくイベントのサブセットを指定できます。

- 他の操作でイベント通知をインターリーブします。

Cisco NX-OS NETCONF エージェントは、アクティブな通知サブスクリプションを持つセッションで、NETCONF 要求を受信し、処理し、応答できます。

機能交換

NETCONF ハンドシェイク中に、Cisco NX-OS NETCONF サーバーは<capabilities> 要素を接続しているNETCONFクライアントに送信して、サーバーが処理できる要求を示します。交換の一部として、サーバーは次の識別子を含めます。これらの識別子は、Cisco NX-OS NETCONF サーバーが通知とインターリーブの両方をサポートしていることをクライアントに通知します。

通知の機能識別子：

```
urn:ietf:params:netconf:capability:notification:1.0
```

インターリーブの機能識別子：

```
urn:ietf:params:netconf:capability:interleave:1.0
```

イベント ストリームの検出

クライアントは、使用可能なすべての <streams> に対し NETCONF の <get> 操作を使用して、Cisco NX-OS NETCONF サーバーがサポートしているストリームを検出できます。Cisco NX-OS は NETCONF ストリームのみをサポートします。イベント ストリームの検出は、要求と応答のシーケンスによって行われます。

使用可能なストリームを取得するための要求：

任意のNETCONFクライアントは、すべてのサポートされているストリームを識別するため、<streams> に対するフィルタとともに、NETCONF <get> 要求を送信できます。次の例は、クライアント要求メッセージのペイロードを示しています。

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get>
      <filter type="subtree">
        <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
           <streams/>
        </netconf>
      </filter>
     </get>
</rpc>
```

応答：

Cisco NX-OS NETCONF サーバーは、クライアントがサブスクライブできる、使用可能なすべてのイベント ストリームで応答します。Cisco NX-OS は NETCONF ストリームのみをサポートします。

```
<rpc-reply message-id="101"
```

```
                    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <data>
        <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
         <streams>
            <stream>
               <name>NETCONF</name>
               <description>default NETCONF event stream </description>
            </stream>
         </streams>
        </netconf>
      </data>
</rpc-reply>
```

サブスクリプションの作成

NETCONF クライアントは、<create-subscription> プロトコル動作により、RPC を通してスイッチ上でイベントのサブスクリプションを作成できます。Cisco NX-OS NETCONF サーバーが <ok/> 要素でのみ応答した場合、サブスクリプションはアクティブです。

同期式の Get および Set 操作とは異なり、サブスクリプションは永続的な非同期操作です。サブスクリプションは、クライアントが明示的にサブスクリプションを閉じるか、セッションがオフラインになるまでアクティブなままです。たとえば、スイッチが再起動すると、セッションはオフラインになります。

クライアントがイベント通知をサブスクライブしていたものの、オフラインになった場合には、サーバーはサブスクリプションを終了し、セッションを閉じます。

サブスクリプションが閉じられた場合、すべてのイベント通知を受信するには、NETCONF クライアントで再接続してサブスクリプションを再度作成する必要があります。

サーバーはサブスクリプションを開始しないため、ユーザーが <create-subscription> 操作を含むクライアントプログラムを書く必要があります。次は、NETCONF クライアントが送信する <create-subscription> の例です。

```
<create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <stream>NETCONF</stream>
    <filter xmlns:ns1="urn:ietf:params:xml:ns:netconf:base:1.0" type="subtree">
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <phys-items>
                    <PhysIf-list>
                        <id>eth1/54/1</id>
                        <phys-items>
                          <operSt/>
                        </phys-items>
                    </PhysIf-list>
                </phys-items>
            </intf-items>
        </System>
      </filter>
</create-subscription>
```

<create-subscription> 操作は、次のオプションのどれでもサポートします。

- <stream>：クライアントがサブスクライブするイベントのストリームを指定します。ストリームを指定しなかった場合、NETCONF ストリーム内のイベントがデフォルトでクライアントに送信されます。
- <filter>：イベントをフィルタリングして、ストリームでイベントのサブセットだけが伝送されるようにできます。

Cisco NX-OS NETCONF サーバーは、サブスクリプションを正常に作成できた場合、応答で <ok> メッセージを返します。

次に、クライアントが <create-subscription> 要求をサーバーに送信し、成功応答を受信した例を示します。

クライアントが受信した、<create-subscription> への応答：

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="urn:uuid:6ff0bda6-d3f1-4288-9a7e-0f30581e4bab">
    <ok/>
</rpc-reply>
```

（注） リプレイを使用したサブスクリプションはサポートされていないため、[開始時間（Start Time）] および [終了時間（Stop Time）] オプションは使用できません。

受信通知

NETCONF クライアントがサブスクリプションを正常に作成すると、Cisco NX-OS NETCONF サーバは、スイッチ内のすべてのイベントについて、使用されたフィルタに関連するイベント通知の送信を開始します。イベント通知は、notification 要素を含む独自の XML フォーマットのドキュメントです。

次に、クライアントが DeviceYang モデルからインターフェイス operSt にサブスクライブしていて、イーサネット インターフェイスがダウンした場合の通知の例を示します。<create-subscription> は、[サブスクリプションの作成（Creating Subscriptions）] セクションにあります。

```
<?xml version="1.0" encoding="UTF-8"?>
<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2020-05-05T10:22:52.260+00:00</eventTime>
    <operation>modified</operation>
    <event>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <phys-items>
                    <PhysIf-list>
                        <id>eth1/54/1</id>
                        <phys-items>
                            <operSt>down</operSt>
                        </phys-items>
                    </PhysIf-list>
                </phys-items>
            </intf-items>
```

```
        </System>
    </event>
</notification>
```

<notification> メッセージには次のフィールドが含まれます。

- <eventTime> はイベントが発生した日時を示すタイムスタンプです。
- <operation> はモデル ノードのイベントのタイプです。
- <event> はクライアントがサブスクライブしているモデル データです。

サブスクリプションの終了

サブスクリプションは、NETCONF クライアントが NETCONF メッセージのペイロードで Cisco NX-OS NETCONF サーバーに特定の操作を送信すると終了します。サブスクリプションの終了は、次のいずれかの方法で発生します。

- サブスクリプション セッションを閉じる。<close-session> 操作は、特定のサブスクリプション セッションに関連して NETCONF サーバーに送信されます。
- NETCONF セッションを終了する。<kill-session> 操作が NETCONF サーバーに送信されます。

すべてのサブスクリプションは、1 つの NETCONF セッションに関連付けられます。これは 1 対 1 の関係です。

NETCONF の例

（注） このセクションのすべての例では、ncclient python ライブラリを使用します。

ncclient を使用した Cisco NX-OS の接続

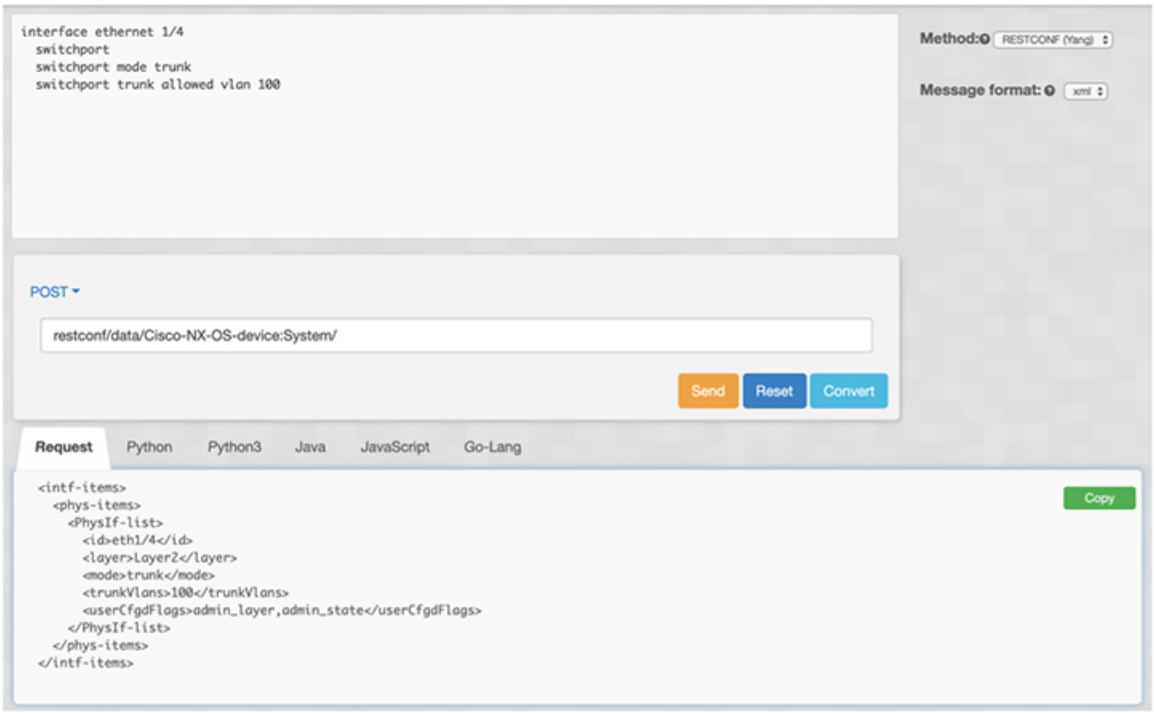
ncclient は、NETCONF クライアント用の Python ライブラリです。次に、ncclient Manager API から Cisco NX-OS への接続を確立する方法の例を示します。

```
device = {
    "address": "10.10.10.10",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}
with manager.connect(host = device["address"],
                         port = device["netconf_port"],
                         username = device["username"],
                         password = device["password"],
                         hostkey_verify = False) as m:
    # do your stuff
```

サンドボックスを使用した NETCONF ペイロードの生成

有効にするには、「NXAPI 開発者サンドボックス」のセクションを参照してください。NETCONF のペイロードを生成するには、メソッドを RESTCONF（Yang）に変更し、メッセージフォーマットを XML に変更します。変換する必要があるコマンドをテキストウィンドウに入力し、**[変換（Convert）]** をクリックすると、同等のペイロードが [要求（Request）] テキストボックスに表示されます。

図 *1 : NCCLIENT*



Cisco NX-OS からの構成データの取得

次に、ncclient を使用して Cisco NX-OS から BGP 設定を取得する方法の例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco!"
}


# create a main() method
def main():
    bgp_dom = """
    <filter type="subtree">
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
                <bgp-items>
                    <inst-items>
```

```
                        <dom-items>
                            <Dom-list/>
                        </dom-items>
                    </inst-items>
                </bgp-items>
            </System>
        </filter>
        """

        with manager.connect(host=device["address"],
                             port=device["netconf_port"],
                             username=device["username"],
                             password=device["password"],
                             hostkey_verify=False) as m:

            # Collect the NETCONF response
            netconf_response = m.get_config(source='running', filter=bgp_dom)
            # Parse the XML and print the data
            xml_data = netconf_response.data_ele
            print(etree.tostring(xml_data, pretty_print=True).decode("utf-8"))


if __name__ == '__main__':
    sys.exit(main())
```

Cisco NX-OS からの実行構成および運用データの取得

次に、Cisco NX-OS 上のすべての物理インターフェイスのインターフェイス カウンタを取得する例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():

    intf_ctr_filter = """
    <filter>
        <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
            <intf-items>
                <phys-items>
                <PhysIf-list>
                    <dbgIfIn-items/>
                    <dbgIfOut-items/>
                </PhysIf-list>
                </phys-items>
            </intf-items>
        </System>
    </filter>"""

    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
```

```
                        password=device["password"],
                        hostkey_verify=False) as m:

        # Collect the NETCONF response
        netconf_response = m.get(filter=intf_ctr_filter)
        # Parse the XML and print the data
        xml_data = netconf_response.data_ele
        print(etree.tostring(xml_data, pretty_print=True).decode("utf-8"))

if __name__ == '__main__':
    sys.exit(main())
```

NETCONF を使用した新しい構成の作成

次に、ncclient の edit config を使用して、名前付きの VLAN 100 を作成する方法の例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():
    add_vlan = """
    <config>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <bd-items>
          <bd-items>
            <BD-list>
              <fabEncap>vlan-100</fabEncap>
              <name>inb_mgmt</name>
            </BD-list>
          </bd-items>
        </bd-items>
      </System>
    </config>
    """


    with manager.connect(host=device["address"],
                        port=device["netconf_port"],
                        username=device["username"],
                        password=device["password"],
                        hostkey_verify=False) as m:

        # create vlan with edit_config
        netconf_response = m.edit_config(target="running", config=add_vlan)
        print(netconf_response)

if __name__ == '__main__':
    sys.exit(main())
```

NETCONF を使用した構成の削除

次に、Cisco NX-OS からループバック インターフェイスを削除する例を示します。

```
from ncclient import manager
import sys
from lxml import etree

device = {
    "address": "nexus",
    "netconf_port": 830,
    "username": "admin",
    "password": "cisco"
}


def main():
    remove_loopback = """
    <config>
      <System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
        <intf-items>
          <lb-items>
            <LbRtdIf-list operation="delete">
              <id>lo10</id>
            </LbRtdIf-list>
          </lb-items>
        </intf-items>
      </System>
    </config>"""


    with manager.connect(host=device["address"],
                         port=device["netconf_port"],
                         username=device["username"],
                         password=device["password"],
                         hostkey_verify=False) as m:

        # create vlan with edit_config
        netconf_response = m.edit_config(target="running", config=remove_loopback)

        print(netconf_response)

if __name__ == '__main__':
    sys.exit(main())
```

NETCONF エージェントのトラブルシューティング

接続のトラブルシューティング

- クライアント システムから、スイッチの管理ポートに ping を実行して、スイッチが到達可能であることを確認します。
- Cisco NX-OS で、**show feature | inc netconf** コマンドを入力してエージェントのステータスを確認します。
- XML 管理インターフェイス（xmlagent とも呼ばれる）というものがあります。これは NETCONF エージェントとはまったく異なりますが、よく混同されます。サーバーが正し

い NETCONF メッセージで応答しない場合は、正しいポート 830 に接続していて、サーバーから正しい <hello> メッセージ（「NETCONF セッションの確立」セクションに示されているものと同様）を受信していることを確認します。

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。