自分のコンピュータを設定する方法

自分のコンピュータでこのラボの作業を行うには、Putty や Terminal などの SSH クライアントが必要です。また、ネットワーク プログラマビリティのラボにアクセスできる必要もあります。

このラボの作業を行うには、事前設定済みの dCloud ラボを使用するか、自分のコンピュータを設定する必要があります。詳細については、「Pre-Event Preparation（イベント前の準備）」および「Lab Setup（ラボの設定）」の各モジュールを確認してください。

デバイス レベルの NETCONF の概要

このラーニング ラボでは、NETCONF プロトコルの基本と、このプロトコルとオープン デバイス プログラマビリティとの関係について学びます。Python コードを使用してラボ環境内の CSR1000V を操作します（デバイスの NETCONF API を使用して、インターフェイスの設定を取得します）。この作業を完了すると、NETCONF のメリットを理解し、NETCONF API を使用してネットワーキング デバイスを操作する基本的な方法を習得することができます。

目標

所要時間：30 分

- NETCONF およびオープン デバイス プログラマビリティのメリットについて理解する。
- NETCONF プロトコルの基本を理解する。
- NETCONF API の使用方法の基本を理解する。
- Python を使用してデバイスの NETCONF サブシステムに接続し、デバイスの設定を取得する。

前提条件

- このラボでは、dCloud ラボ環境を使用します。自分のローカル ラップトップまたは Ubuntu ホストを使用してこのラボの演習を行う場合は、「Lab Setup（ラボ設定）」および「Pre-Event Preparation（イベント前の準備）」の各モジュールを見直してしてください。

背景情報

- Python に詳しくない場合には、「REST APIs and Python（REST API および Python）」モジュールを必ず確認してください。このモジュールでは、Python の基本について説明しています。

DMI NETCONF API をサポートするデバイスへアクセスする

- これらのラボの演習には、必ず CSR1000V を使用してください。このラボでは、NETCONF と連携する DMI エージェントを使用できます。

Python

- サンプル コードを実行するには、使用しているコンピュータに Python がインストールされている必要があります。

仮想環境

- このラボでは、既存の Python 環境で生じる問題を回避するために、*仮想環境*を使用して `ncclient` バージョン 0.5.2 を実行します。
- このステップを完了する方法については、「`Lab Setup`(**ラボの設定**)」および「`Pre-Event Preparation`(**イベント前の準備**)」モジュールを参照してください。

開発ライブラリ

- 自分のコンピュータで作業する場合は、Python、`libxml2`、および `libxslt` などの開発バージョンを使用することをお勧めします。
- このステップを完了する方法については、「`Lab Setup`(**ラボの設定**)」および「`Pre-Event Preparation`(**イベント前の準備**)」モジュールを参照してください。

Git リポジトリを複製する

- 自分のコンピュータで作業する場合は、Git リポジトリを複製してください。
- このステップを完了する方法については、「`Lab Setup`(**ラボの設定**)」および「`Pre-Event Preparation`(**イベント前の準備**)」モジュールを参照してください。

作業ディレクトリへサンプルをコピーする

Git リポジトリからサンプルを、`virtualenv` を使用してセットアップしたディレクトリにコピーすることができます。Linux または Mac では、以下のコマンドを使用してください。

```
$ pwd
$ /home/user/Code/mycode
$ cp ../devnet-express-code-samples/module06/* .
```

複製されたディレクトリから、関連するコード サンプルとサンプル ファイルをコピーします。この操作では、`mycode` ディレクトリおよび `devnet-express-code-samples` ディレクトリが同じディレクトリに配置されていることを前提としています(この場合は、`/home/user/Code`)。

ただし、一部のコード サンプルでは、別の場所にあるファイルを含めることを前提にしている場合があります。このような場合は、これらの別のファイルを見つけることができるように、スクリプトを調整する必要があります。

ステップ 1：NETCONF およびオープン デバイス プログラマビリティのメリットについて理解する

NETCONF は、YANG データ モデリング言語を使用したモデル化が行われている場合に、各種のベンダーやオペレーティング システムのすべてに対して、一貫性がありプログラム可能なインターフェイスを提供します。YANG については、次のラーニング ラボで詳細に取り上げます。現時点では、一貫性のある NETCONF API が YANG により実現するということを覚えておいてください。また、NETCONF はネットワークおよびデバイスの自動化をシンプル化するのに役立ちます。ネットワークおよびネットワーク デバイスの自動化については、以下に示すようなさまざまな課題が存在します。

1. プラットフォーム間の不整合
    - ネットワーク デバイスのベンダーが異なったり、実行されているオペレーティング システムが異なると、使用されているプログラム インターフェイスも異なることが多い。
2. 非構造化データ
    - コマンドライン インターフェイス（CLI）はネットワーク デバイスを操作する際の最もよく知られたインターフェイスだが、CLI は構造化されておらず、プログラムによる操作が難しい。
3. 検証の課題に伴う困難
    - 設定の変更をデバイスに適用する場合、変更が正しく行われたかどうかを確認するためにはカスタムの検証手順が必要。
4. 設定の変更を展開する際の不確実性
    - 設定の変更はこれまで SNMP か「スクリーンスクレイピング」を使用して行われることが多かったが、どちらの手法も信頼性が低く、エラーが発生しやすい。

これらの問題の例を実際に確認するために、以下の Python のコード例を確認してください。この例では、Nexus9KV 設定のホスト名が解析されています。

```
#!/usr/bin/env python

import re
import sys

def main():
    """
    Open a file called sandbox-nexus9kv-config.txt.
    Print each line that matches a regular expression for a hostname.
    """
    HOSTNAME = ''
```

```
    NXOS_HOSTNAME_REGEX = '^hostname (.*)$'
    NXOS_DOMAIN_REGEX = '^ip domain-name (.*)$'
    with open('sandbox-nexus9kv-config.txt', 'r') as nexus_config:
        for line in nexus_config:
            if re.match(NXOS_HOSTNAME_REGEX, line):
                HOSTNAME = re.search(NXOS_HOSTNAME_REGEX, line).group(1) +
'.'
            elif re.match(NXOS_DOMAIN_REGEX, line):
                HOSTNAME += re.search(NXOS_DOMAIN_REGEX, line).group(1)

    print(HOSTNAME)

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

この sandbox-nexus9kv-config.txt ファイルには、NEXUS9KV の CLI 設定が含まれています。上記の Python スクリプトでは、デバイスの設定ファイルに関連する行が表示されている場合に、デバイスのホスト名を出力することが想定されます。

このスクリプトを実行すると、Python スクリプトが設定の各行を解析することを確認できます。特定の行が IOS コマンド ip domain-name <STRING> または hostname <STRING> に一致する場合、この行はシンタックスの <STRING> を解析してホスト名を作成します。

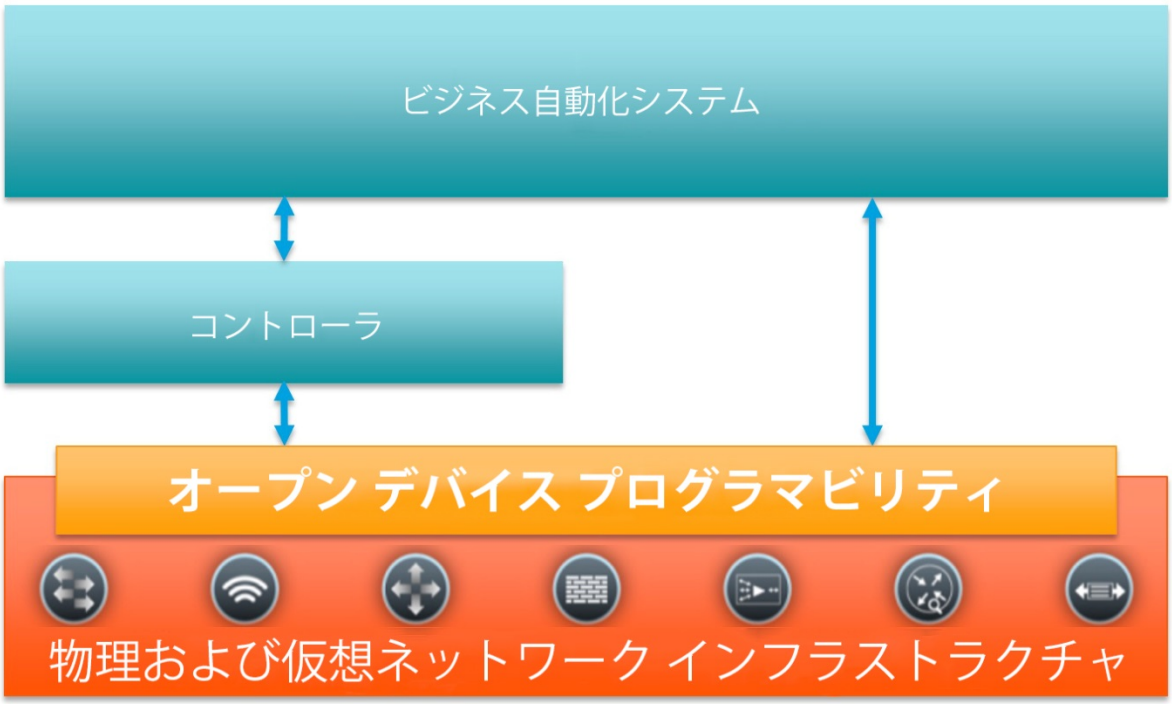
実際のスクリプトの例を以下に示します。この実習は自習によって行うことができます。当該のコードは、devnet-express-code-samples/module06/ フォルダにあります。

```
$ python3 screen_scraping-1.py
n9kvswitchfcs.cisco.com
$
```

上記の例は、従来の CLI を解析して、ネットワークを自動化しようとした場合に発生する、いくつかの課題を示しています。まず、CLI が非構造化データを提供している点です。CLI を解析するには、各シナリオ用にカスタムの正規表現を作成して、テストする必要があります。次に、スクリプトやアプリケーションで適切なデータを取得できるようにするためには、CLI の詳細な知識が必要である点です。

NETCONF では、構造化されたデータ（XML）および YANG データ モデリング言語を使用して、共通の設定シンタックスおよび operational 状態をモデル化することによって、適切な YANG データ モデルをサポートする各種のプラットフォームで API の一貫性を確保します。YANG について混乱していても、慌てないでください。次のラボですぐに説明します。

NETCONF は、ネットワーク デバイスへのプログラムによるアクセスをシンプル化するための、一連のオープン デバイス プログラマビリティ技術の 1 つにすぎないことに留意してください。他の例には、RESTCONF などがあります。下記の図では、ネットワーク オペレーティング システムのスタック全体で使用されている、オープン デバイス プログラマビリティ技術について説明しています。



オープン デバイス プログラマビリティを利用することで、アプリケーションの開発者やネットワーク エンジニアは、プログラム可能なインターフェイスに対してコードを作成することができます。コントローラおよび商用の自動化システムでも、operational データの取得や構成管理など、各種のユースケースで同様の API を使用することができます。

ラーニング ラボの次のパートでは、NETCONF、SNMP、RESTCONF、および REST など、当該分野の各種技術の間の関係について、概要を説明します。

ステップ 2：RESTCONF、SNMP、NETCONF、および REST の関係を理解する

現在、ネットワーク デバイスでは多数のプログラマチック インターフェイスが使用されています。デバイスの中には、NETCONF、REST、および RESTCONF API のすべてを備えているものもあります（または、同じ API を複数実装している場合もあります）。以下では、デバイスのプログラマビリティの歴史を紹介して、各種の API バインディングの関係と、これらのすべてが同じデバイス上で使用される可能性がある理由について説明します。

歴史的な視点：SNMP

まず、ネットワーク デバイスは従来、シンプル ネットワーク管理プロトコル（SNMP）を使用して管理されてきました。ただし、ネットワーク オペレータのコミュニティでは、一般的に、SNMP では対応できない多数のギャップが存在することが確認されていました。これらの情報について詳しくは、RFC3535 を参照してください。SNMP に関する問題の例として、複数デバイスに対して設定の変更を行う際に、トランザクション上の制約を実現できないという点があります。たとえば、オーケストレーション ソフトウェアで複数デバイスに対して設定の変更を展開する必要がある場合、1 つのデバイスで変更が失敗しても、SNMP は他のネットワーク要素に対してプッシュされた変更をロールバックすることができません。

また、SNMP MIB はそう理解しやすいものとはいえません。JSON など、シリアル化される他のデータ記述言語に比べ、MIB は判読が困難です。その結果、多くのエンジニアは分かりやすいツールである CLI を利用しています。

NETCONF

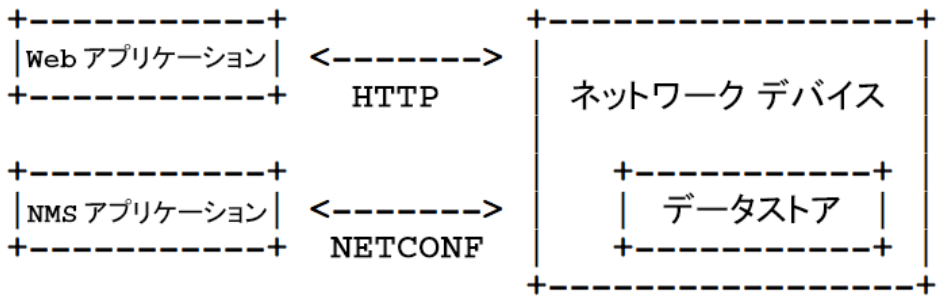
RFC3535 で特定された問題に対処するために、NETCONF プロトコルが作成されました。NETCONF では一般的に、安全な送受信のために SSH が使用され、データの交換には XML が使用されます。NETCONF には、プログラムを使用してネットワーク デバイスを操作するために、リモート プロシージャ コール（RPC）が用意されています。また、NETCONF プロトコルによって操作される設定およびデータは、通常は YANG データ モデリング言語を使ってモデル化されます。ただし、実装によっては、YANG より古いレガシーの NETCONF 実装が使用されている場合もあります。NETCONF API が YANG を使用してモデル化されているかどうかを確認することが重要です。この方法については、この後のラーニング ラボで説明します。

RESTCONF

RESTCONF プロトコルは NETCONF の後に開発されており、ネットワーク デバイスに対する REST ライクなインターフェイスを提供します。このプロトコルでは、転送のために HTTP（S）が使用され、YANG でモデル化されたデータを使用するネットワーク デバイスを操作するのに JSON または XML が使用されます。RESTCONF と NETCONF はどちらも YANG によってモ

デル化された同じデータにアクセスできますが、使用する API バインディングは異なります。RESTCONF は NETCONF の代替にはならないことに注意してください。RESTCONF は、YANG を使用してモデル化された同じデータ（設定データや operational データなど）を使用するための別のメカニズムを提供するだけです。プログラマーや設計者は、どちらの API を使用するかを、自分の経験、スキルセット、および要件に基づいて選択できます。

たとえば、下記の図は RESTCONF RFC の抜粋で、NETCONF および RESTCONF API の両方を同時に使用する方法を示しています。



例としては、NETCONF は、NETCONF を利用した既存のコードがある NMS（またはオーケストレータ）で使用し、顧客の既存のコードおよびスキルセットの関係で、Web アプリケーションでは RESTCONF を使用します。

こうした場合でも慌てる必要はありません。次のラーニング ラボでは、YANG について、さらに詳しく説明します。

REST

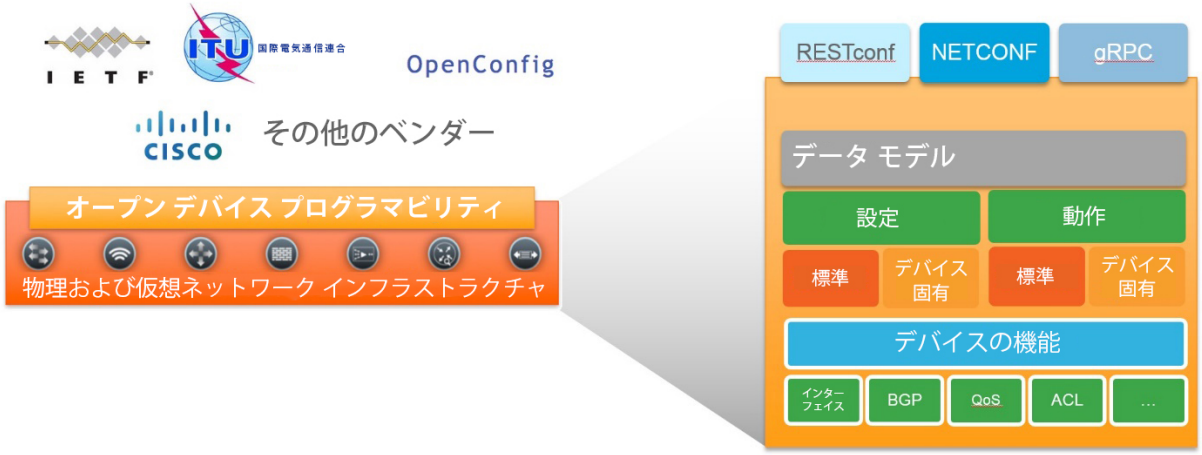
さらに、ネットワーク デバイスでは REST インターフェイスが使用されている場合もあります。RESTCONF や NETCONF とは異なり、REST API がモデル化される方法はプラットフォーム、オペレーティング システム、ベンダー、API を設計したエンジニアによって大きく異なります。同様に、REST API も実装によって大きく異なります。

gRPC

gRPC もメッセージング フレームワークであり、XML や JSON の代わりにプロトコル バッファを使用して、データをシリアル化します。gRPC は NETCONF や RESTCONF によって使用されているものと同じデータストアにアクセスすることができます。gRPC は、比較的最近追加された、オープン デバイス プログラマビリティのフレームワークです。

オープン デバイス プログラマビリティとの関係

オープン デバイス プログラマビリティは、OpenConfig および IETF などの組織が策定する一般的な YANG データ モデルに対してプログラマビリティを実現する、一連の技術です。これらのインターフェイスには、以下の図に示す NETCONF、RESTCONF、および gRPC が含まれます。



YANG データ モデルと NETCONF の関連に進む前に、NETCONF の基本的な事項の一部を把握しましょう。ラーニング ラボの次のパートでは、NETCONF API の機能について詳しく説明します。

ステップ 3：開始前に NETCONF について知っておくべきこと

NETCONF API を利用するには、開始前に以下の情報について理解しておく必要があります。この情報は、デバイスの参照資料または YANG データ モデル（該当する場合）で確認できます。この情報を念頭に置いて、NETCONF の利用を開始してください。

- **NETCONF サーバ**
  - NETCONF サーバは、一般にはルータやスイッチなどのネットワーク デバイスです（オーケストレータやコントローラの場合もあります）。
  - NETCONF クライアントは、NETCONF プロトコルを使用して RPC（リモート プロシージャ コール）を NETCONF サーバに対して発行し、デバイスの設定や操作を行います。
- **RPC**
  - NETCONF プロトコルでは、NETCONF サーバ（ネットワーク デバイス、オーケストレータ、またはコントローラ）に対する operations を実行するために、RPC が使用されます。
  - このラーニング ラボは、NETCONF を使用したネットワーク デバイスの操作を中心に扱います。
  - NETCONF RPC では一般に、operations を実行するためのトランスポートとして SSH が使用され、データのフォーマッティングとシリアル化に XML が使用されます。
- **Operations**
  - Operations は NETCONF サーバまたはデバイスを設定したり、操作したりするための固有のメソッドまたはコマンドです。
  - Operations は XML でエンコードされます。
  - NETCONF サーバで一般にサポートされる基本的な operations の例は以下のとおりです。
    - get：operational データを取得する
    - get-config：設定データを取得する
    - edit-config：デバイス設定を編集する
    - copy-config：設定を他のデータ ストア（不揮発性メモリなど）にコピーする
    - delete-config：データ ストア内の設定を削除する
  - ベンダーがデバイス固有の operations を公開している場合もあります。
  - ネットワーク エンジニアおよび開発者は、実装された YANG モデルまたはデバイスあるいはプラットフォームの API に関するドキュメンテーションから、NETCONF サーバがサポートする operations を理解する必要があります。
- **データ ストア**
  - NETCONF サーバは、データ ストアを使用して設定および operational データ（状態）を保存します。
  - NETCONF operations は、データストアの設定データを更新したり、取得したりします（つまり、読み取りと書き込みを行います）。

    - NETCONF operations は、データストアから operational データも取得します（読み取ります）。
    - NETCONF は、operational データと設定データを明確に分離して、混乱を防ぐため、読み取り専用データと読み取りおよび書き込み可能なデータの区別を徹底しています。
- メッセージ
    - Operations、および Operations から返されるデータは XML でエンコードされます。
    - NETCONF クライアントおよびサーバでは、メッセージのフォーマットおよびデータのやり取りに XML が使用されます。

認証についての留意事項

NETCONF API では、一般に、送受信に SSH が使用されます。NETCONF セッションでも SSH に対して同じ認証メソッドが使用されます。ネットワーク エンジニアや開発者は、認証にユーザ名/パスワードまたは SSH キーを使用できます。

NETCONF の仕組み

下記の図は、NETCONF の仕組みについて説明しています。この図は、**get** operation を使用して、プログラムで operational データを取得する NETCONF RPC の例に必要な手順を示しています。

1. この NETCONF クライアントは、NETCONF サーバに対して SSH セッションを確立します。

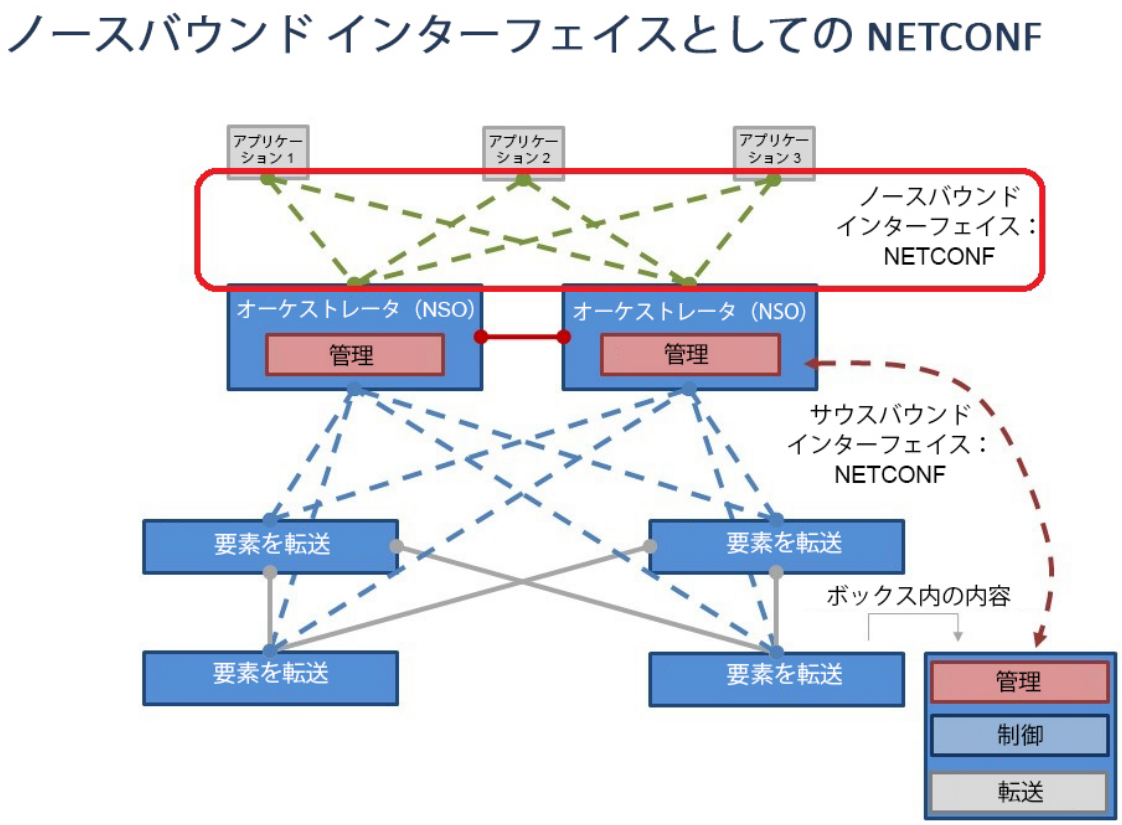
2. NETCONF クライアントとサーバでは、capabilities のやり取りをするために、NETCONF の **hello** メッセージをやり取りします。

3. **hello** メッセージを交換することで、クライアントは RPC を発行することができるようになります。このシナリオでは、このクライアントは **get** operation を送信し、サーバが operational データを返します。特定のデータに対しては、**get** operational がフィルタ処理されることに注意してください。フィルタは XML を使用して作成されます。

これで NETCONF の仕組みの概要を把握しました。次は、NETCONF API のさまざまな面について説明します。

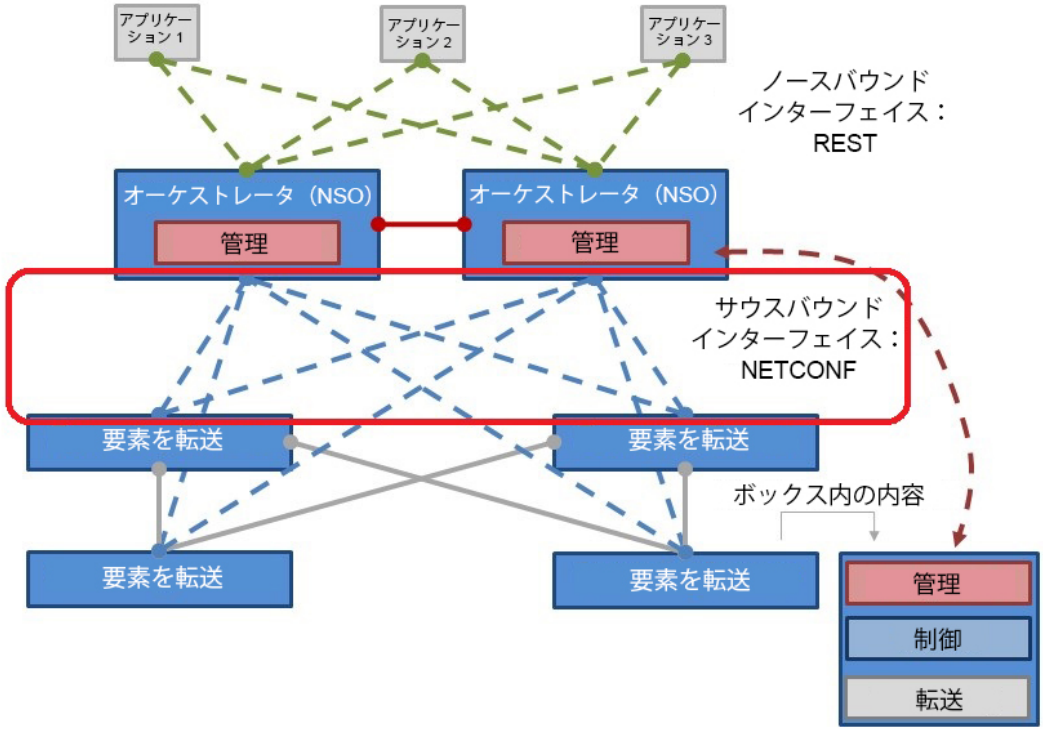
ステップ 4：NETCONF API について理解する

NETCONF API を操作するには複数の方法があります。シスコのネットワーク サービス オーケストレータ（NSO）など、一部のコントローラやオーケストレータは、ノースバウンド NETCONF インターフェイスを使用します。このような形で NETCONF をノースバウンド インターフェイス（NBI）として使用することにより、ユーザは複数のデバイスを抽象性の高いレイヤで設定したり、集中管理されたプラットフォームからネットワーク全体の operational データを収集したりすることができます。この場合、コントローラまたはオーケストレータは、必要なサウスバウンド インターフェイス（NETCONF や SSH など）を使用して、ネットワーク インフラを適宜操作します。



また、次に示すように、個別のネットワーク デバイスでも NETCONF インターフェイスが直接使用されます。エンジニアは、OpenConfig などのプロジェクトを使用することで、YANG データ モデリング言語を使用して共通の設定をモデル化します。デバイスで OpenConfig YANG モデルがサポートされていれば、ネットワークのオペレーティング システムやベンダーを問わずに、複数のデバイスで API の一貫性が確保されます。そのため、ネットワーク エンジニアやソフトウェア エンジニアはどちらも、一貫した NETCONF API を基にアプリケーションやスクリプトを作成することができます。次に示す例では、オーケストレータは、ベンダー製品、シンプルな Python スクリプト、または複雑なカスタム アプリケーションです。

サウスバウンド インターフェイスとしての NETCONF



このラーニング ラボの次のパートでは、上記の図のような形で、Python を使用して、ネットワーク デバイスに対する最初の NETCONF API コールを行います。

ステップ 5：NETCONF の実行：実際に試してみる

Cisco CSR1000V の例：NETCONF over SSH で接続する

それでは、実際の NETCONF の動作について詳しく見ていきましょう。

このラボでは dCloud ラボ環境を使用してトポロジを実行しています。このトポロジについて確認が必要な場合は、モジュール 00 およびモジュール 03 を見直してください。

始めに、Python スクリプトと NETCONF を使用して CSR1000V デバイスに接続する、基本的な例を実行します。デバイスへの接続が完了すると、Python スクリプトは CSR1000V（NETCONF サーバ）によってサポートされる NETCONF capabilities を出力します。このスクリプトは、前のモジュールで複製した `devnet-express-code-samples/module06` にあり、`get_hostname.py` という名前です。

```
#!/usr/bin/env python

# import the ncclient library
from ncclient import manager
import sys
import xml.dom.minidom


# the variables below assume the user is leveraging the
# dCloud lab environment.
#
# use the IP address or hostname of your CSR1000V device
HOST = '198.18.133.218'
# use the NETCONF port for your CSR1000V device
PORT = 2022
# use the user credentials for your CSR1000V device
USER = 'admin'
PASS = 'C1sco12345'


# create a main() method
def main():
    """
    Main method that retrieves the hostname from config via NETCONF.
    """
    with manager.connect(host=HOST, port=PORT, username=USER,
                         password=PASS, hostkey_verify=False,
                         device_params={'name': 'default'},
                         allow_agent=False, look_for_keys=False) as m:

        # XML filter to issue with the get operation
        hostname_filter = '''
                          <filter>
                              <native xmlns="urn:ios">
                                  <hostname></hostname>
                              </native>
```

```
                        </filter>
                        '''
        result = m.get_config('running', hostname_filter)
        xml_doc = xml.dom.minidom.parseString(result.xml)
        hostname = xml_doc.getElementsByTagName("hostname")
        print(hostname[0].firstChild.nodeValue)


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

では、このコードの動作を見てみましょう。

- まず、`ncclient` および `sys` ライブラリをインポートします。
  - *自分のコンピュータで作業している場合は*、このコードを実行する前に、NCClient ライブラリをインストールする必要があります。
  - Python3 との互換性を確保するために、最低でも `ncclient` バージョン 0.5.2 を使用する必要があります。
- 次の行は複数の名前を作成します。これらの名前は、環境に合わせて必要に応じて更新できます。
  - ラボ環境で CSR1000V を使用している場合は、何も変更する必要はありません。
  - このコードを実行する前に、環境に接続していることを確認してください。
  - 必要に応じてモジュール 03 の手順を確認してください。
  - このコード スニペットは、以下のように、必要な引数を使用して NETCONF over SSH セッションを作成します（空白は読みやすさのために追加されています）。
  - `  with manager.connect(host=HOST, port=PORT, username=USER,`
  - `                       password=PASS, hostkey_verify=False,`
  - `                       device_params={'name': 'default'},`
  - `                       allow_agent=False, look_for_keys=False) as m:`
  - `host` = リモート デバイスの IP アドレスまたはホスト名。
  - `port` = SSH セッション用の NETCONF ポート。
  - `username` = SSH セッションを認証するためのユーザ名。
  - `password` = SSH セッションを認証するためのパスワード。
  - `hostkey_verify` = ~/.ssh/known_hosts からホストキーを無効にします。
  - `device_params` = ベンダー固有の operations を有効にします（この例では特殊な operations は行いません）。
  - `look_for_keys` = ユーザ名/パスワードを使用しているため、公開キーを無効にします。
  - `allow_agent` = ユーザ名/パスワードを使用しているため、公開キーを無効にします。
  - `with ... as` 文を使用することにより、ランタイム時に何らかの例外が発生した場合、セッションは正常に終了されます。

- 次に、SSH セッションが確立されたら、`hostname_filter` という文字列を作成します。この文字列には、NETCONF からの応答をフィルタ処理するための XML データが含まれています。この例では、デバイスの設定からホスト名を取得するだけです。
- 次の行のコードでは、最初の NETCONF operation が呼び出されます。ここでは `<get_config>` 操作を発行します。この操作は、この操作では、running configuration を指定して、デバイスの設定からホスト名のみを取得するようにフィルタを使用します。
- 次の 2 行は、文字列としてエンコードされている、返された XML を解析するためのメソッドを呼び出します。これで、この XML ドキュメントからホスト名を取得できます。
- 最後に、`if __name__ == '__main__':` という式があります。
    - これにより、スクリプトが直接呼び出された場合（モジュールとしてインポートされた場合ではなく）にのみ、`main()` メソッドが実行されるようになります。

セキュリティに関する考慮事項

このラーニング ラボの例では、公開キーを使用するのではなく、ユーザ名/パスワードを Python スクリプトに直接挿入しています。

```
USER = 'admin'
PASS = 'C1sco12345'
```

さらに、このラーニング ラボの例では、`hostkey_verify=False` を使用して、`known_hosts` ファイルを無視しています。

```
with manager.connect(host=HOST, port=PORT, username=USER, password=PASS,
                     hostkey_verify=False, device_params={'name': 'default'},
                     look_for_keys=False, allow_agent=False) as m:
```

実稼働環境では、このような設定を**使用しないでください**。

これらの設定と全体的なアプローチは、ラボのテスト環境の簡単な構築と実行に便利です。

このサンプル コードを実行するには、次のようにします。

1. サンプル コードがあるディレクトリに移動します。このディレクトリは、`devnet-express-code-samples/module06/` です。
2. 次に、以下のようにしてこのスクリプトを実行します。
    - *Windows の場合*:`py3 get_hostname.py`
    - *Mac OS または Linux の場合*:`python3 get_hostname.py`

次のような結果が表示されます。

```
$ python3 get_hostname.py
csr-nwp1
$
```

いかがでしょう。これで、NETCONF を使用してデバイスが接続され、hello メッセージと capabilities のやり取りが完了しました。次に、running configuration のある NETCONF データストアにクエリし、デバイスのホスト名を取得しました。

YANG データ モデルを中心にオープン デバイス プログラマビリティを取り上げた、前のラーニング ラボの手順を確認してください。次のラーニング ラボでは、YANG と NETCONF の関係について説明します。