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、Python および Requests ライブラリをインストールする必要があります。

Python のインストール

- 使用しているオペレーティング システム用の Python 3.4.2 以降をインストールします。
- https://www.python.org/downloads/ からインストーラをダウンロードします。

Requests ライブラリのインストール

- Requests ライブラリの詳細を確認するには、ここをクリックします。
- Mac OS に Python 3 の Requests をインストールするには、次のコマンドライン コマンドを使用します。
  - ```
    sudo curl -O https://raw.githubusercontent.com/pypa/pip/master/contrib/get-pip.py
    sudo python3 get-pip.py
    pip3 install requests
    ```
- Windows に Requests をインストールするには、以下を行います。
  - pip は Windows 用の Python 3.4 に組み込まれています。
  - pip が Python 3 のパスに存在するかどうかを確認します。

    ```
    C:\>pip -V
    pip 1.5.6 from C:\Python34\lib\site-packages
    ```
  - pip が Python34 のパスに存在する場合は、次のコマンドを実行します。

    ```
    pip install requests
    ```

Flask ライブラリのインストール

- Flask ライブラリの詳細を確認するには、ここをクリックします。
- Mac OS に Python 3 の Flask をインストールするには、次のコマンド ラインを使用します。
  - ```
    pip3 install flask
    ```
- Windows に Flask をインストールするには、以下を行います。
  - pip が Python34 のパスに存在する場合は、次のコマンドを実行します。

    ```
    pip install flask
    ```

- また、APIC-EM コントローラにアクセスできる必要があります。

    - 常時接続可能な APIC-EM ラボの DevNet Sandbox は次の URL から利用できます: https://sandboxapic.cisco.com。
    - SSL 証明書を必ず受け入れるようにしてください。

Git Repo を複製する

- コーディング スキルのサンプル コードを複製します。

```
git clone https://github.com/CiscoDevNet/coding-skills-sample-code
```

コーディング 102（初級 2）：Python から REST API をコールする

このラーニング ラボでは、Python から REST API をコールする基本的な方法について学習します。

目標

所要時間：35 分

- Python から REST API をコールする方法について理解する
- REST API から返された JSON を解析して使用する方法を確認する

前提条件

背景

- このラボを開始する前に『Coding 101 Rest Basics Learning Lab（コーディング 101：Rest の基本に関するラーニング ラボ）』を完了しておくことをお勧めします。

APIC-EM コントローラへのアクセス

- これらのサンプル コードを実行するためには、APIC-EM コントローラにアクセスできる必要があります。
- **自分の APIC-EM コントローラを使用していない場合は、**DevNet サンドボックスの Always-On APIC-EM ラボ（https://sandboxapic.cisco.com/）を使用します。

Python

- サンプル コードを実行するには、使用しているマシンに Python 3 がインストールされている必要があります。
- Python をインストールする方法については、この Web ページ上部の「**How to Setup Your Own Computer（自分のコンピュータを設定する方法）**」の項を参照してください。

Python のリクエスト ライブラリ

- これらのサンプル コードでは、Python のリクエスト ライブラリを使用して REST API コールをシンプル化しています。
- リクエスト ライブラリをインストールする方法については、この Web ページ上部の「**How to Setup Your Own Computer（自分のコンピュータを設定する方法）**」の項を参照してください。

Python の Flask ライブラリ

- ステップ 6 では、NeXt UI と連携するために Python の Flask ライブラリを使用します。
- Flask ライブラリをインストールする方法については、この Web ページ上部の「**How to Setup Your Own Computer（自分のコンピュータを設定する方法）**」の項を参照してください。

Git Repo を複製する

- デスクトップでコマンド ターミナルを開きます。
- `cd \` と入力して、ルート ディレクトリに移動します。
- `mkdir DevNetCode\<your-name>` と入力して、「C:\DevNetCode\yourname」というディレクトリを作成します。
    - 例：`mkdir DevNetCode\brTiller`

- cd \DevNetCode\<your-name> と入力して、新しいディレクトリに移動します。
    - 例：cd \DevNetCode\brTiller
- GitHub からコーディング スキルのサンプル コードを複製します。次のコマンドを入力します。

```
git clone https://github.com/CiscoDevNet/coding-skills-sample-code
```

```
C:\DevNetCode\brTiller>git clone https://github.com/CiscoDevNet/coding-skills-sample-code
Cloning into 'coding-skills-sample-code'...
remote: Counting objects: 148, done.
remote: Total 148 (delta 0), reused 0 (delta 0), pack-reused 148Receiving objects:  60% (89/148)
Receiving objects: 100% (148/148), 41.09 KiB | 0 bytes/s, done.
Resolving deltas: 100% (78/78), done.
Checking connectivity... done.
```

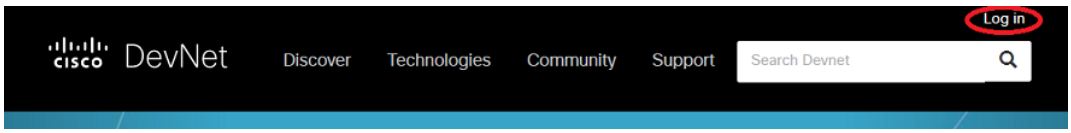
作成したディレクトリに「coding-skills-sample-code」ディレクトリが確認できるはずです。

```
C:\DevNetCode\brTiller>dir
 Volume in drive C is System
 Volume Serial Number is 808B-CEEA

 Directory of C:\DevNetCode\brTiller

12/15/2015  03:16 PM    <DIR>          .
12/15/2015  03:16 PM    <DIR>          ..
12/15/2015  03:16 PM    <DIR>          coding-skills-sample-code
               0 File(s)              0 bytes
               3 Dir(s)  55,545,159,680 bytes free
```

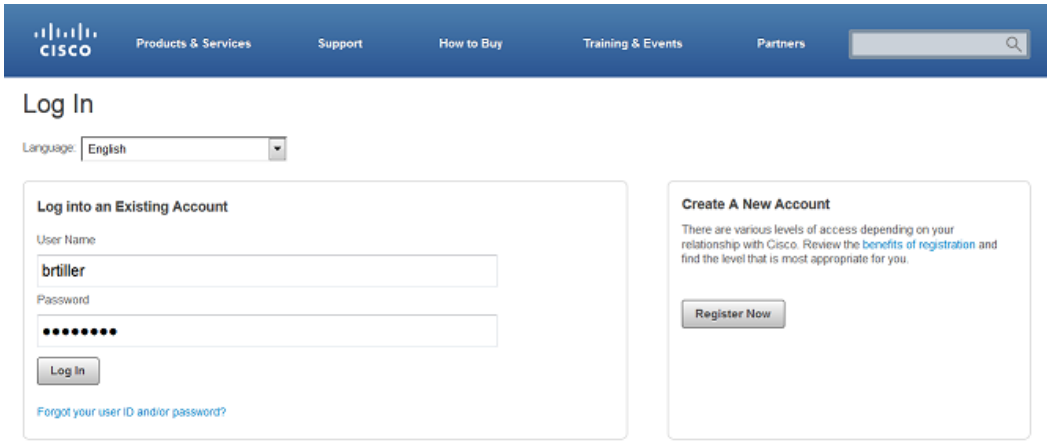
ステップ 1：APIC-EM API リソースの確認

このラボでは、REST API の例として APIC-EM API を使用します。そのため、ラボの開始前に DevNet で APIC-EM リソースを確認しておく必要があります。

1. ブラウザで、DevNet にアクセスします。
    - Web ページの右上にある [ログイン（Login）] リンクをクリックします。



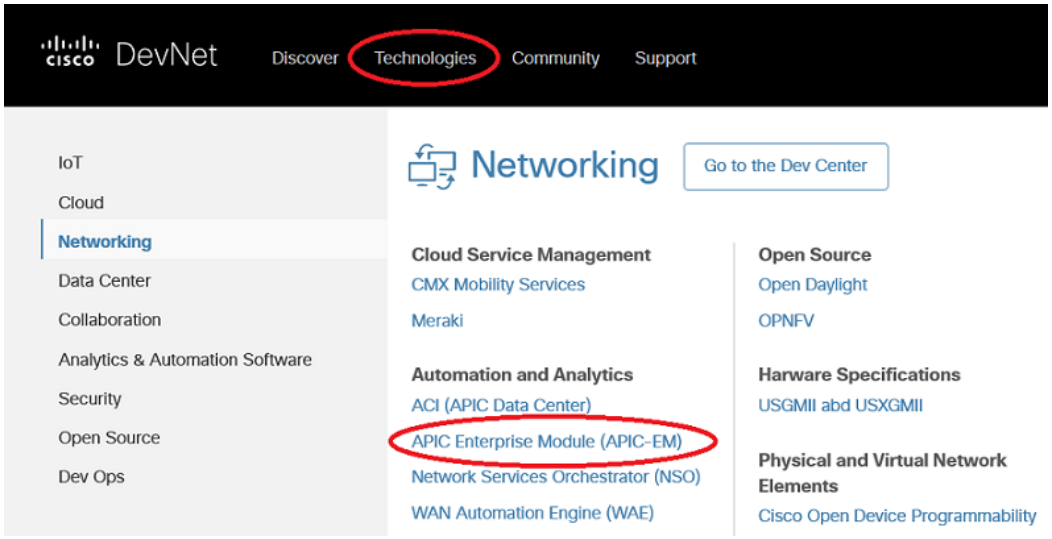
- ログイン Web ページで自分のログイン クレデンシャル（CCO ID）を入力し、[ログイン（Log In）] ボタンをクリックします。



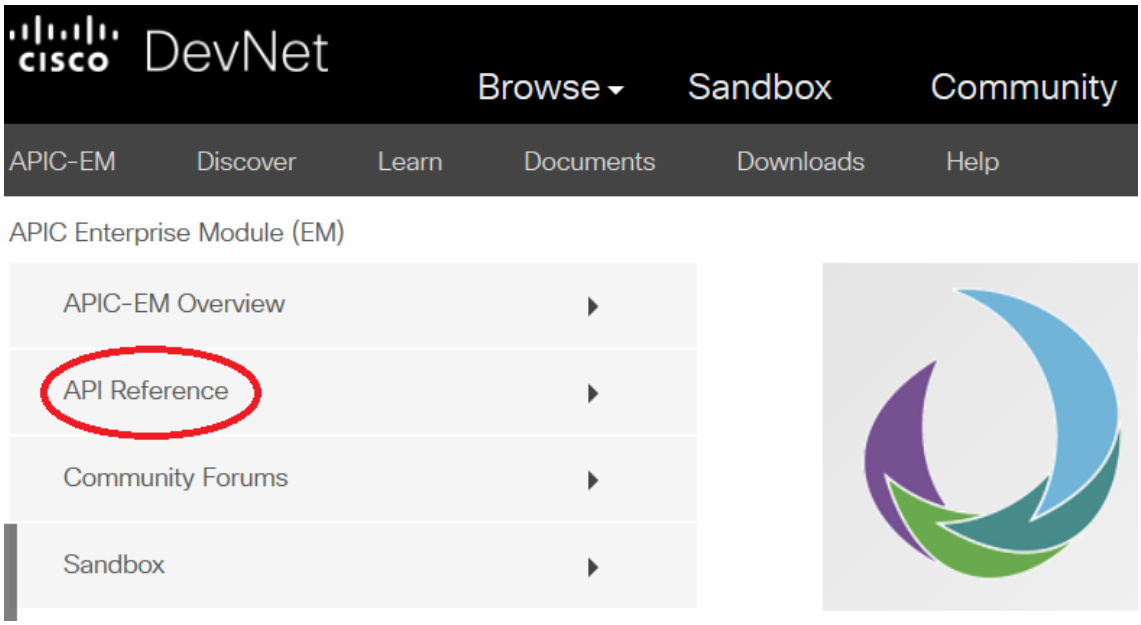
  - Cisco Connection Online ID（CCO ID）を持っていない場合は、[今すぐ登録（Register Now）] ボタンをクリックして、指示に従います。登録が完了したら、ログイン Web ページに戻ってクレデンシャルを入力します。

2. トップにあるメニューを使用して、APIC-EM 開発者リソースに直接アクセスします。

- [テクノロジー（Technologies）] リンクをクリックして、[テクノロジー（Technologies）] メニューにアクセスします。メニューで、[ネットワーキング（Networking）] をクリックした後、[APIC-EM] をクリックします



3. これで、APIC-EM の Web ポータルにログインしました。ラボでの実習中には、API リファレンス ドキュメントは、別のタブで開いてください。



ステップ 2：Python から最初の REST コールを発信する

次のコードを利用して、リクエスト ライブラリを使用したシンプルな POST リクエストを実行します。

```
# import requests library
import requests

#import json library
import json

# put the ip address or dns of your apic-em controller in this url
url = 'https://{APIC-EM-Controller}/api/v1/ticket'

#the username and password to access the APIC-EM Controller
payload = {"username":"devnetuser","password":"Cisco123!"}

#Content type must be included in the header
header = {"content-type": "application/json"}

#Performs a POST on the specified url.
response= requests.post(url,data=json.dumps(payload), headers=header, verify=False)

# print the json that is returned
print(response.text)
```

では、このコードの動作を見てみましょう。

- まず、リクエスト ライブラリをインポートします。
  - リクエスト ライブラリのインストールは、ステップ 1 で指示に従って完了している必要があります。
  - リクエスト ライブラリの関数は HTTP コールをシンプルにします。また、post、get、delete などの REST コールをサポートし、REST 機能に対応します。
- 次に、JSON ライブラリをインポートします。
  - このコード スニペットでは、JSON ライブラリを使用して、データを JSON 形式にします。
  - このライブラリは Python 内にありますが、ライブラリの関数にアクセスするにはこのライブラリをインポートする必要があります。
- 次の行は、このリクエストで使用する URL が含まれる **url** という変数を作成します。
  - この例では、サービス チケットを返す REST コールを使用します。このチケットは、APIC-EM API でのすべての REST コールの認証に使用されます。
  - REST コールの仕組みについては、『Coding 101: Rest Basics Learning Lab（コーディング 101：REST の基本についてのラーニング ラボ）』を参照してください。
- *payload = {"username":"devnetuser","password":"Cisco123!"}*
  - サービス チケットを作成するには、プログラムが APIC-EM コントローラにログインする必要があるため、ユーザ名とパスワードが必要です。指定するユーザ名およびパスワードは、サンドボックスの Always-On Database for APIC-EM（https://sandboxapic.cisco.com）のものです。
- *header = {"content-type": "application/json"}*
  - content-type ヘッダーは、APIC-EM コントローラがどのような形式でフォーマットされたデータを受け取るかを示します。
- *response= requests.post(url,data=json.dumps(payload), headers=header, verify=False)*
  - POST メソッドを使用して、指定された URL へのリクエストを実行します。
  - ペイロード データが JSON 形式に変換されて渡されます。
  - ヘッダーが渡されます。
  - 関数からのレスポンスが、**response** という変数に返されます。
- *print (response.text)* コマンドが、レスポンスで返された JSON を画面に出力します。

SSL 証明書に関する注意と verify=False の使用

このラーニング ラボの例では **verify=False** パラメータが使用されています。

```
# this statement performs a POST on the specified url
response= requests.post(url,data=json.dumps(payload), headers=header, verify=False)
```

verify=False を設定すると、リクエスト ライブラリで SSL 証明書の検証が無効になります。

この設定は、自己署名証明書が使用されていることが多いラボのテスト環境には便利です。

実稼働環境ではこの設定を**使用しないでください。**

実際に確認

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. ターミナルを開き、ステップ 1 で作成した **DevNetCode\<your-name>** という名前のディレクトリに移動します。
    - ステップ 1 の「**前提条件**」の項では、ソース コード ファイルを Git リポジトリからこのディレクトリに複製しました。作成したディレクトリに、サブディレクトリ **coding-skills-sample-code** が確認できるはずです。このサブディレクトリがない場合は、ステップ 1 の前提条件の項に戻り、手順に従ってディレクトリを作成し、Git リポジトリを複製します。
2. **coding102-REST-python-ga** ディレクトリに移動します。ターミナルで **cd \DevNetCode\<your-name>\coding-skills-sample-code\coding102-REST-python-ga** と入力します。
3. **create-ticket.py** ファイルを開きます。たとえば、Windows では **notepad create-ticket.py** と入力します。
4. URL の {APIC-EM-Server} 部分の IP アドレスを、使用しているコントローラの IP アドレスに変更します。
    - *自分の APIC-EM コントローラを使用していない場合は、*DevNet サンドボックスの Always-On APIC-EM ラボを使用します（sandboxapic.cisco.com と入力する）。
5. ファイルを保存します。エンコーディング タイプがオプションの場合は、**UTF-8** を選択します。
6. Python コマンドを入力後、コマンド プロンプトにファイル名を入力して、リターン キーを押します。
    - *Windows では、***py -3 create-ticket.py** と入力するか、**python create-ticket.py** と入力します。
    - *Mac OS または Linux の場合は、***python3 create-ticket.py** と入力します。
7. プログラムが実行されるか、エラー メッセージが表示されます。

次のような結果が表示されます。

```
C:\DevNetCode\brTiller\coding-skills-sample-code\coding102-REST-python-ga>python create-ticket.py
C:\Python34\lib\site-packages\requests\packages\urllib3\connectionpool.py:789: InsecureRequestWarnin
g: Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See:
 https://urllib3.readthedocs.org/en/latest/security.html
  InsecureRequestWarning)
{"response":{"serviceTicket":"ST-501-mYBRUPmvpbEWwgFddjTd-cas"},"version":"1.0"}
```

このプログラムは、サービス チケット データから JSON をそのまま表示します。

確認事項

- セキュリティ警告が表示されますか。
- サービス チケットの値が表示されますか。

次のいくつかの項では、他の REST API コールを発信する場合にサービス チケットを使用する方法や、レスポンスで返された JSON を解析する方法について説明します。

ステップ 3：ネットワークに関する情報を取得する

先ほどのシンプルな例を利用し、レスポンスで返された JSON を解析して使用する方法について学びます。

これらのファイルは、使用しているワークステーションの /DevNetCode/<your-name>/coding-skills-sample-code/coding102-REST-python-ga/ ディレクトリにあります。

- **create-ticket.py**：サービス チケットの作成例。ステップ 2 で使用します。
- **get-network-hosts.py**：サービス チケットのレスポンスを解析し、JSON データの pretty プリントを出力してホストのリストを表示する最初のアプリケーションです。
- **get-network-devices.py**：ネットワーク デバイスのリストを取得し、JSON を解析して networkDeviceId の値を表示します。
- **build-topology.py**：デバイスおよびインターフェイスを取得する方法、およびスプレッドシート形式のテキスト トポロジを構築して表示する方法を示します。
- **build-topology-web-server.py**：デバイスおよびインターフェイスを取得する方法、およびグラフィカルなトポロジを構築する方法を示します。

get-network-hosts.py

このサンプル コードでは REST コールを使用して、ネットワーク ホストのリストを取得します。コーディング 101 では、ホストはワークステーションなどのエンド デバイスであり、ネットワーク ケーブルでスイッチなどのネットワーク デバイスに接続されたり、ワイヤレス デバイスに接続されたりする、ということを説明しました。ここでの目的は、ホストを見つけ、その情報を表示することです。

```
# import requests library
import requests

#import json library
import json

#variable to hold access to the controller
controller='sandboxapic.cisco.com'

# Create the service ticket URL
url = "https://" + controller + "/api/v1/ticket"

#the username and password to access the APIC-EM Controller
payload = {"username":"devnetuser","password":"Cisco123!"}

#Content type must be included in the header
header = {"content-type": "application/json"}

#Performs a POST on the specified url to get the service ticket
response= requests.post(url,data=json.dumps(payload), headers=header, verify=False)

#convert response to json format
r_json=response.json()

#parse the json to get the service ticket
ticket = r_json["response"]["serviceTicket"]

# URL for Host REST API call to get list of exisitng hosts on the network.
url = "https://" + controller + "/api/v1/host"

#Content type must be included in the header as well as the service ticket
header = {"content-type": "application/json", "X-Auth-Token":ticket}

# this statement performs a GET on the specified host url
response = requests.get(url, headers=header, verify=False)
```

```
# json.dumps serializes the json into a string and allows us to
# print the response in a 'pretty' format with indentation etc.
print ("Hosts = ")
print (json.dumps(response.json(), indent=4, separators=(',', ': ')))
```

では、このコードの動作を見てみましょう。ここでは、主要なコードの変更に重点を置きます。

- *controller='sandboxapic.cisco.com'*
  - **controller** という名前の変数に APIC-EM コントローラの IP または DNS で引ける名前を割り当てます。ここで割り当てられる名前は、APIC-EM Always-On サンドボックスのものです。
- *url = "https://" + controller + "/api/v1/ticket"*
  - これらの文字列を連結して、サービス チケットの URL を作成します。
- *r_json=response.json()*
  - 返されたデータを JSON 形式に変換して、この文字列にアクセスできるようにします。
- *ticket = r_json["response"]["serviceTicket"]*
  - レスポンス データにアクセスして、サービス チケットの情報を取得します。
- *header = {"content-type": "application/json", "X-Auth-Token":ticket}*
  - API 認証のために「X-Auth-Token」としてサービス チケットを渡します。
- *print (json.dumps(response.json(), indent=4, separators=(',', ': ')))*
  - pretty プリントの出力は、可読性が高いです。

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. **coding102-REST-python-ga** ディレクトリに移動します。ターミナルで **cd \DevNetCode\<your-name>\coding-skills-sample-code\coding102-REST-python-ga** と入力します。
2. **controller** 変数に、APIC-EM コントローラの IP または DNS で引ける名前を割り当てます。
   - **get-network-hosts.py** ファイルを開きます。たとえば、Windows では **notepad get-network-hosts.py** と入力します。
   - *自分の APIC-EM コントローラを使用していない場合は、*DevNet サンドボックスの Always-On APIC-EM ラボを使用します（sandboxapic.cisco.com）。
     - controller='sandboxapic.cisco.com'
3. ファイルを保存します。エンコーディング タイプがオプションの場合は、**UTF-8** を選択します。
4. Python コマンドを入力後、コマンド プロンプトにファイル名を入力して、リターン キーを押します。
   - *Windows では、***py -3 get-network-hosts.py** と入力するか、**python get-network-hosts.py** と入力します。
   - *Mac OS または Linux の場合は、***python3 get-network-hosts.py** と入力します。
5. プログラムが実行されるか、エラー メッセージが表示されます。

   次のような結果が表示されます。

確認事項

- 各ネットワーク ホストの ID は確認できますか。
- 「get-network-hosts.py」ファイルで、ホスト URL に「?limit=1&offset=1」を追加して *url = "https://" + controller + "/api/v1/host?limit=1&offset=1"* とし、コードを再度実行します。どこが変わりましたか。またその理由は何ですか。

次の項では、ネットワーク デバイスの取得方法と、ソース コードを読みやすくする方法について学習します。

ステップ 4：ネットワーク デバイスを取得する

このステップでは、コードをモジュール化し、ネットワーク デバイスを取得するための関数を作成します。また、JSON のレスポンス データを解析して、返された各ネットワーク デバイスの ID を出力します。

get-network-devices.py

このサンプル コードではネットワーク デバイスの REST コールを使用して、ネットワーク デバイスのリストを取得します。ネットワーク デバイスは、ワークステーションやデバイスがデータやその他のリソースを共有することができるようにネットワークに接続するのに使用されるルータ、スイッチ、ハブなどのコンポーネントです。ここでの目的は、ネットワーク デバイスを見つけ、その情報を表示することです。

```
# import requests library
import requests

#import json library
import json

controller='sandboxapic.cisco.com'

def getTicket():
  # put the ip address or dns of your apic-em controller in this url
  url = "https://" + controller + "/api/v1/ticket"

  #the username and password to access the APIC-EM Controller
  payload = {"username":"devnetuser","password":"Cisco123!"}

  #Content type must be included in the header
  header = {"content-type": "application/json"}

  #Performs a POST on the specified url to get the service ticket
  response= requests.post(url,data=json.dumps(payload), headers=header, verify=False)

  #convert response to json format
  r_json=response.json()

  #parse the json to get the service ticket
  ticket = r_json["response"]["serviceTicket"]

  return ticket


def getNetworkDevices(ticket):
  # URL for network device REST API call to get list of existing devices on the network.
  url = "https://" + controller + "/api/v1/network-device"

  #Content type must be included in the header as well as the ticket
  header = {"content-type": "application/json", "X-Auth-Token":ticket}

  # this statement performs a GET on the specified network-device url
  response = requests.get(url, headers=header, verify=False)

  # json.dumps serializes the json into a string and allows us to
  # print the response in a 'pretty' format with indentation etc.
  print ("Network Devices = ")
  print (json.dumps(response.json(), indent=4, separators=(',', ': ')))

  #convert data to json format.
  r_json=response.json()

  #Iterate through network device data and print the id and series name of each device
    for i in r_json["response"]:
```

```
    print(i["id"] + " " + i["series"])

#call the functions
theTicket=getTicket()
getNetworkDevices(theTicket)
```

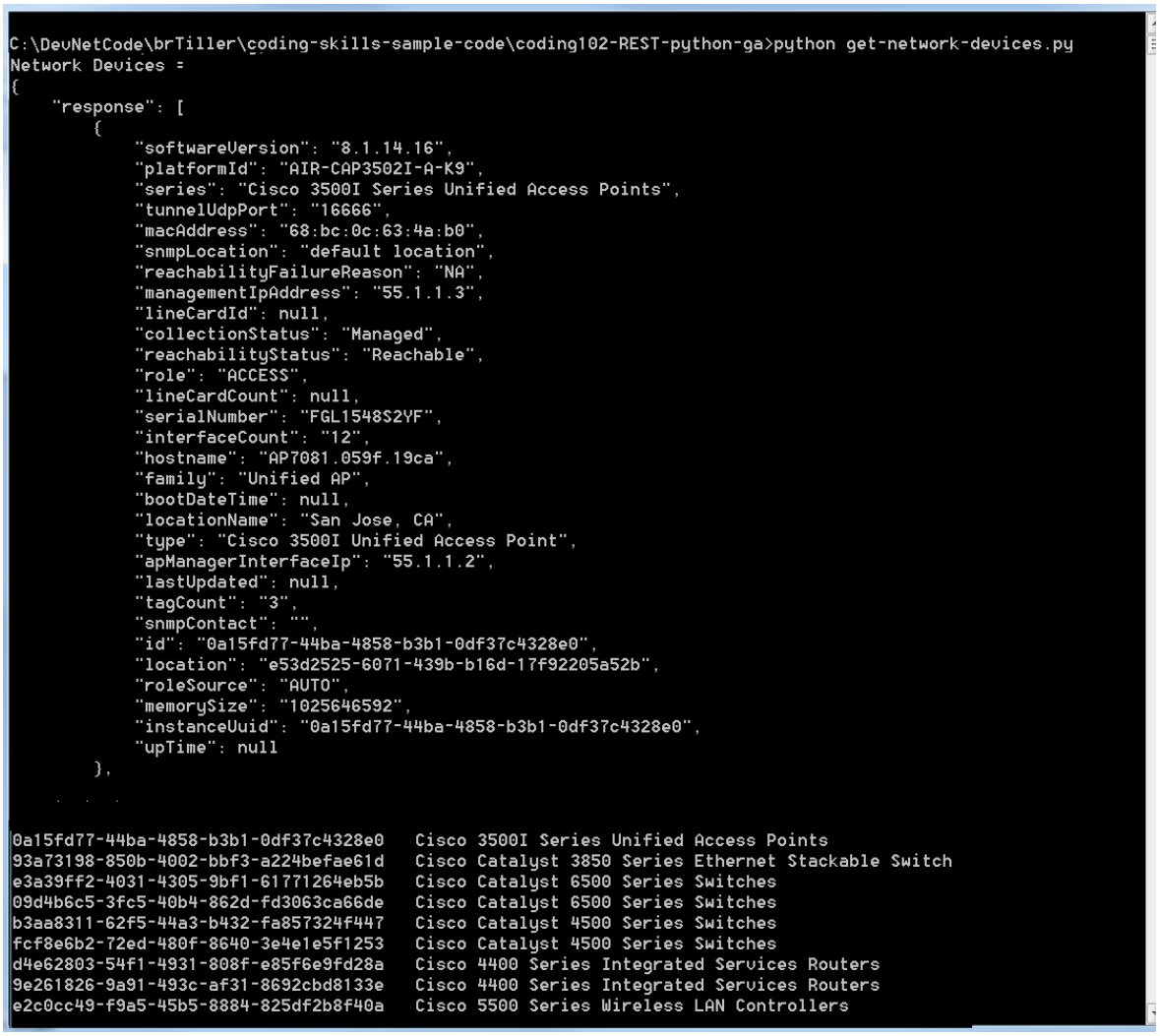
では、このコードの動作を見てみましょう。ここでは、主要なコードの変更に重点を置きます。

- *def getTicket():*
  - getTicket という名前の関数を定義します。この関数はサービス チケットを作成し、そのチケットを返します。
- *def getNetworkDevices(ticket):*
  - getNetworkDevices という、チケット パラメータを使用する関数を定義します。このチケット パラメータにはサービス チケットのデータが含まれます。この関数は、ネットワーク デバイスを返します。
- *url = "https://" + controller + "/api/v1/network-device"*
  - これらの文字列を連結して、ネットワーク デバイスを取得するための URL を作成します。
- *header = {"content-type": "application/json", "X-Auth-Token":ticket}*
  - API 認証のために「X-Auth-Token」としてチケット データが指定された HTTP ヘッダーです。
- *for i in r_json["response"]: print(i["id"] + " " + i["series"])*
  - ネットワーク データの JSON データを解析して、デバイスの ID およびシリーズ名を出力します。
- *theTicket=getTicket()*
  - getTicket() 関数をコールして、サービス チケット データを「theTicket」変数に割り当てます。
- *getNetworkDevices(theTicket)*
  - getNetworkDevices(theTicket) 関数をコールして、「theTicket」変数のサービス チケット データを渡します。

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. **coding102-REST-python-ga** ディレクトリに移動します。ターミナルで **cd \DevNetCode\<your-name>\coding-skills-sample-code\coding102-REST-python-ga** と入力します。
2. **controller** 変数に、APIC-EM コントローラの IP または DNS で引ける名前を割り当てます。
   - **get-network-devices.py** ファイルを開きます。たとえば、Windows では **notepad get-network-hosts.py** と入力します。
   - *自分の APIC-EM コントローラを使用していない場合は*、DevNet サンドボックスの Always-On APIC-EM ラボを使用します（sandboxapic.cisco.com）。
     - controller='sandboxapic.cisco.com'
3. ファイルを保存します。エンコーディング タイプがオプションの場合は、**UTF-8** を選択します。
4. Python コマンドを入力後、コマンド プロンプトにファイル名を入力して、リターン キーを押します。
   - *Windows では*、**py -3 get-network-devices.py** と入力するか、**python get-network-devices.py** と入力します。
   - *Mac OS または Linux の場合は*、**python3 get-network-devices.py** と入力します。
5. プログラムが実行されるか、エラー メッセージが表示されます。

   次のような結果が表示されます。簡略化のために、警告や一部のレコードは割愛されています。



確認事項

- getNetworkDevices 関数を編集して、各ネットワーク デバイスの管理 IP アドレスとロケーション名を表示します。
- getHosts(theTicket) という新しい関数を作成して、ネットワーク ホストを取得して表示します。ホスト データを取得するための URL については、ステップ 3 を参照してください。

次の項では、ネットワーク トポロジを構築する方法を学習します。

ステップ 5：ネットワークトポロジを構築する

このステップでは、トポロジを取得してデータを解析し、デバイス同士がどのようにリンクされているか、およびそれらのリンクのステータスを判別して表示します。

build-topology.py

このサンプル コードでは、アプリケーションの REST コールを使用して、ノードと呼ばれるデバイスのリストと、それらを接続するインターフェイスであるリンクのリストを取得します。ここでの目的は、デバイスを見つけ、デバイス自体の情報と、デバイス同士が接続されている方法を表示することです。

```
# import requests library
import requests

#import json library
import json

# Disable warnings
requests.packages.urllib3.disable_warnings()

#controller='sandboxapic.cisco.com'
controller='devnetapi.cisco.com/sandbox/apic_em'


def getTicket():
    # put the ip address or dns of your apic-em controller in this url
    url = "https://" + controller + "/api/v1/ticket"

    #the username and password to access the APIC-EM Controller
    payload = {"username":"devnetuser","password":"Cisco123!"}

    #Content type must be included in the header
    header = {"content-type": "application/json"}

    #Performs a POST on the specified url to get the service ticket
    response= requests.post(url,data=json.dumps(payload), headers=header, verify=False)

    print (response)

    #convert response to json format
    r_json=response.json()

    #parse the json to get the service ticket
    ticket = r_json["response"]["serviceTicket"]

    return ticket



def getTopology(ticket):
    # URL for topology REST API call to get list of existing devices on the network, and build topology
    url = "https://" + controller + "/api/v1/topology/physical-topology"

    #Content type as well as the ticket must be included in the header
    header = {"content-type": "application/json", "X-Auth-Token":ticket}

    # this statement performs a GET on the specified network device url
    response = requests.get(url, headers=header, verify=False)

    # json.dumps serializes the json into a string and allows us to
    # print the response in a 'pretty' format with indentation etc.
    print ("Topology = ")
    print (json.dumps(response.json(), indent=4, separators=(',', ': ')))

    #convert data to json format.
    r_json=response.json()
```

```
    #Iterate through network device data and list the nodes, their interfaces, status and to what they connect
    for n in r_json["response"]["nodes"]:
        print()
        print()
        print('{:30}'.format("Node") + '{:25}'.format("Family") + '{:20}'.format("Label")+ "Management IP")
        if "platformId" in n:
            print('{:30}'.format(n["platformId"]) + '{:25}'.format(n["family"]) + '{:20.14}'.format(n["label"]) + n["ip"])
        else:
            print('{:30}'.format(n["role"]) + '{:25}'.format(n["family"]) + '{:20.14}'.format(n["label"]) + n["ip"])
        found=0    #print header flag
        printed=0  #formatting flag
        for i in r_json["response"]["links"]:
            #check that the source device id for the interface matches the node id.  Means interface originated from this device.
            if i["source"] == n["id"]:
                if found==0:
                    print('{:>20}'.format("Source Interface") + '{:>15}'.format("Target") +'{:>28}'.format("Target Interface") + '{:>15}'.format
("Status") )
                    found=1
                    printed=1
                for n1 in r_json["response"]["nodes"]:
                    #find name of node to which this one connects
                    if i["target"] == n1["id"]:
                        if "startPortName" in i:
                            print("    " + '{:<25}'.format(i["startPortName"]) + '{:<18.14}'.format(n1["label"]) + '{:<25}'.format(i["endPortNam
e"]) + '{:<9}'.format(i["linkStatus"]) )
                        else:
                            print("    " + '{:<25}'.format("unknown") + '{:<18.14}'.format(n1["label"]) + '{:<25}'.format("unknown") + '{:<9}'.f
ormat(i["linkStatus"]) )
                        break;
        found=0
        for i in r_json["response"]["links"]:
            #Find interfaces that link to this one which means this node is the target.
            if i["target"] == n["id"]:
                if found==0:
                    if printed==1:
                        print()
                    print('{:>10}'.format("Source") + '{:>30}'.format("Source Interface") + '{:>25}'.format("Target Interface") + '{:>13}'.forma
t("Status"))
                    found=1
                for n1 in r_json["response"]["nodes"]:
                    #find name of node to that connects to this one
                    if i["source"] == n1["id"]:
                        if "startPortName" in i:
                            print("    " + '{:<20}'.format(n1["label"]) + '{:<25}'.format(i["startPortName"]) + '{:<23}'.format(i["endPortName"]
) + '{:<8}'.format(i["linkStatus"]))
                        else:
                            print("    " + '{:<20}'.format(n1["label"]) + '{:<25}'.format("unknown") + '{:<23}'.format("unknown") + '{:<8}'.form
at(i["linkStatus"]))
                        break;

theTicket=getTicket()
getTopology(theTicket)
```

では、このコードの動作を見てみましょう。ここでは、主要なコードの変更に重点を置きます。

- *def getTopology(theTicket):*
    - getTopology という名前の関数を定義します。この関数はトポロジ データを読み取ります。また、ネットワーク上のデバイスであるノードを解析して、それらのデバイスに関する情報の一部を示します。さらに、ノードを接続するインターフェイスであるリンクのデータも解析して、デバイスの接続方法に関する情報と、リンクのステータスも示します。「theTicket」というパラメータが渡されて、認証のために使用されます。
- *for n in r_json["response"]["nodes"]:*
    - 各ノードのディクショナリ データを n に読み込み、そこからデータを解析します。
- *if "platformId" in n:*
    - n のノード データに「platformId」キーがあるかどうかをチェックします。レコードの中にはこのキーがないものもあり、その場合別のフィールド データを調べる必要があります。
- *if "startPortName" in i:*

- i のインターフェイス データに「startPortName」キーがあるかどうかをチェックします。レコードの中にはこのキーがないものもあり、その場合別のフィールド データを調べる必要があります。

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. **coding102-REST-python-ga** ディレクトリに移動します。ターミナルで **cd \DevNetCode\<your-name>\coding-skills-sample-code\coding102-REST-python-ga** と入力します。
2. **controller** 変数に、APIC-EM コントローラの IP または DNS で引ける名前を割り当てます。
   - **build-topology.py** ファイルを開きます。たとえば、Windows では **notepad build-topology.py** と入力します。
   - *自分の APIC-EM コントローラを使用していない場合は*、DevNet サンドボックスの Always-On APIC-EM ラボ( )を使用します。
     - controller='sandboxapic.cisco.com'
3. ファイルを保存します。エンコーディング タイプがオプションの場合は、**UTF-8** を選択します。
4. Python コマンドを入力後、コマンド プロンプトにファイル名を入力して、リターン キーを押します。
   - *Windows では*、**py -3 build-topology.py** と入力するか、**python build-topology.py** と入力します。
   - *Mac OS または Linux の場合は*、**python3 build-topology.py** と入力します。
5. プログラムが実行されるか、エラー メッセージが表示されます。

   次のような結果が表示されます。簡略化のために、一部のレコードは割愛されています。

```
Node                        Family                  Label               Management IP
ISR4451-X/K9                Routers                 CAMPUS-Router2      210.2.1.1
    Source              Source Interface        Target Interface       Status
    ISR4451-X/K9        GigabitEthernet0/0/2    GigabitEthernet0/0/2   up
    WS-C6503-E          GigabitEthernet1/3      GigabitEthernet0/0/1   up


Node                        Family                  Label               Management IP
WS-C4507R+E                 Switches and Hubs       CAMPUS-Dist1        212.1.10.100
    Source Interface        Target           Target Interface        Status
    GigabitEthernet5/38     AIR-CT5508-K9    GigabitEthernet0/0/1    up
    GigabitEthernet5/48     WS-C4507R+E      GigabitEthernet5/48     up

    Source              Source Interface        Target Interface       Status
    WS-C6503-E          GigabitEthernet1/1      GigabitEthernet5/7     up
    WS-C3850-48U        GigabitEthernet1/0/1    GigabitEthernet5/5     up


Node                        Family                  Label               Management IP
AIR-CT5508-K9               Wireless Controller     Campus-WLC-5508     55.1.1.2
    Source              Source Interface        Target Interface       Status
    WS-C4507R+E         GigabitEthernet5/38     GigabitEthernet0/0/1   up


Node                        Family                  Label               Management IP
WS-C4507R+E                 Switches and Hubs       CAMPUS-Dist2        212.1.20.2
    Source              Source Interface        Target Interface       Status
    WS-C4507R+E         GigabitEthernet5/48     GigabitEthernet5/48    up
    WS-C3850-48U        GigabitEthernet1/0/2    GigabitEthernet5/5     up
    WS-C6503-E          GigabitEthernet1/1      GigabitEthernet5/7     up


Node                        Family                  Label               Management IP
AIR-CAP3502I-A-K9           Unified AP              AP7081.059f.19ca    55.1.1.3
    Source Interface        Target           Target Interface        Status
    GigabitEthernet0        WS-C3850-48U     GigabitEthernet1/0/26   up
```

確認事項

- スクリプトを実行したら、出力されたノード データを確認します。label キーを role や nodeType などの別のキーで置き換えて、ソース コードを変更します。スクリプトを再度実行して、表示されたデータで変更を確認します。

- 出力されたトポロジ データを確認し、ホスト デバイスがいくつ存在しているか、どのデバイスに接続されているかを確認します。
- クラウド ノードから開始して、提供されたトポロジ データを使用してトポロジの最初の 3 つの階層の図を描画します。ヒント：Label 属性で指定されてている、マッチするデバイスを探します。たとえば、クラウド ノードは 4 つのデバイスに接続されており、そのうちの 1 つは Branch-Router1 というデバイスです。トポロジ データの Label フィールドで Branch-Router1 が接続しているデバイスを判別します。他の 3 つのデバイスについても同じ手順を実施します。

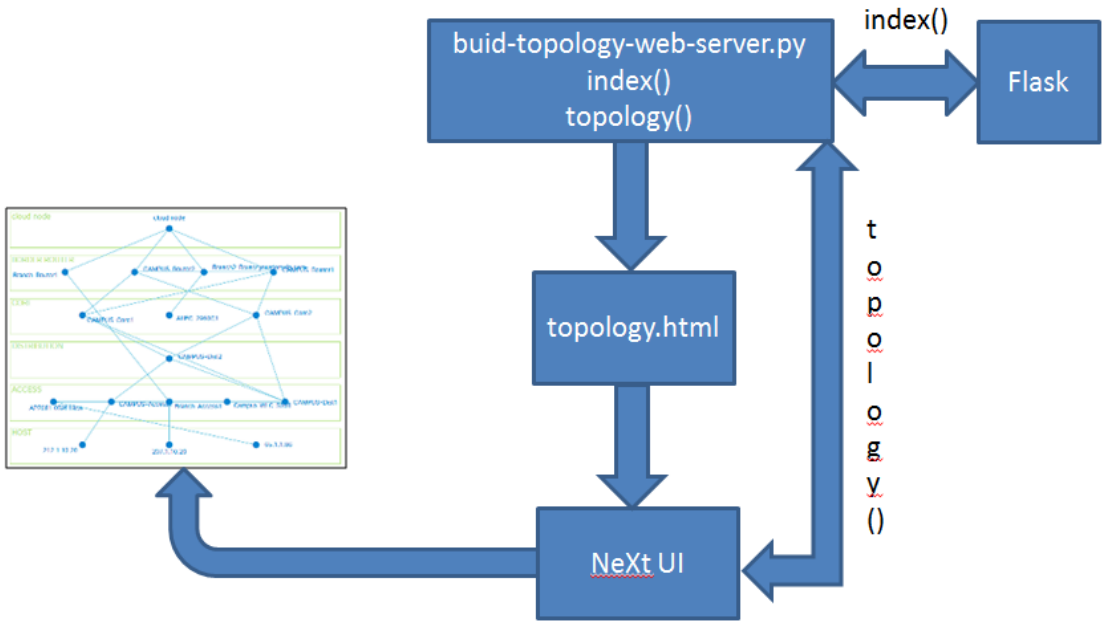
次の項では、NeXt UI ツールキットおよび Flask を使用してネットワーク トポロジを構築し、グラフィカルに表示する方法について学習します。

ステップ 6：ネットワーク トポロジを構築し、グラフィカルに表示する

このステップでは、ネットワーク トポロジのデータを取得し、シスコの NeXt UI という別のツールを使用してそのデータを解析し、トポロジをグラフィカルに表示します。また、Flask という Web サーバ ツールも使用します。このツールは、NeXt UI と連携して使用するのに適しています。Flask は Python モジュールです。

それでは、このプロセスの仕組みの概要を、順を追って確認してから、ソース コードの詳細を確認しましょう。次のフローチャートもご覧ください。

1. Python スクリプト build-topology-web-server.py を開始すると、Flask Web サーバが起動されます。
2. Flask は、build-topology-web-server.py にある index() 関数をコールします。この関数は、一連のイベントを開始します。
3. index() 関数は、topology.html ファイルをロードします。このファイルには、NeXt UI ツールキット ライブラリを使用する javascript が含まれています。
4. NeXt は build-topology-web-server.py ファイルにある topology() 関数をコールして、JSON 形式でトポロジ データを取得します。
5. 次に、NeXt は、このデータからネットワーク トポロジを解析してレンダリングします。



それでは、ソース コードを詳しく見てみましょう。ここでは、Python スクリプトにのみ焦点を当てます。

build-topology-web-server.py

このサンプル コードは Flask Web アプリケーションを起動し、ノードと呼ばれるデバイスおよびそれらのデバイスを接続するインターフェイスであるリンクのリストを取得するためにコールされる関数を提供します。このスクリプトと、Flask および NeXt UI は、ネットワーク トポロジを読み取り、このトポロジを Web ページとしてグラフィカルに表示します。

```
# import requests library
import requests

# import json library
import json

# import flask web framework
from flask import Flask

# from flask import render template function
from flask import render_template, jsonify controller = 'sandboxapic.cisco.com'


def getTicket():
  # put the ip address or dns of your apic-em controller in this url
  url = "https://" + controller + "/api/v1/ticket"

  # the username and password to access the APIC-EM Controller
  payload = {"username": "devnetuser", "password": "Cisco123!"}

  # Content type must be included in the header
  header = {"content-type": "application/json"}

  # Performs a POST on the specified url to get the service ticket
  response = requests.post(url, data=json.dumps(payload), headers=header, verify=False)

  print(response)

  # convert response to json format
  r_json = response.json()

  # parse the json to get the service ticket
  ticket = r_json["response"]["serviceTicket"]

  return ticket


def getTopology(ticket):
  # URL for network-device REST API call to get list of exisiting devices on the network.
  url = "https://" + controller + "/api/v1/topology/physical-topology"

  # Content type as well as the ticket must be included in the header
  header = {"content-type": "application/json", "X-Auth-Token": ticket}

  # this statement performs a GET on the specified network device url
  response = requests.get(url, headers=header, verify=False)

  # convert data to json format.
  r_json = response.json()

  # return json object
  return r_json["response"]


# intialize a web app
app = Flask(__name__)


# define index route to return topology.html
@app.route("/")
def index():
  # when called '/' which is the default index page, render the template 'topology.html'
```

```
  return render_template("topology.html")


# define an reset api to get topology data
@app.route("/api/topology")
  def topology():
    # get ticket
    theTicket = getTicket()

    # get topology data and return `jsonify` string to request
    return jsonify(getTopology(theTicket))


if __name__ == "__main__":
    app.run()
```

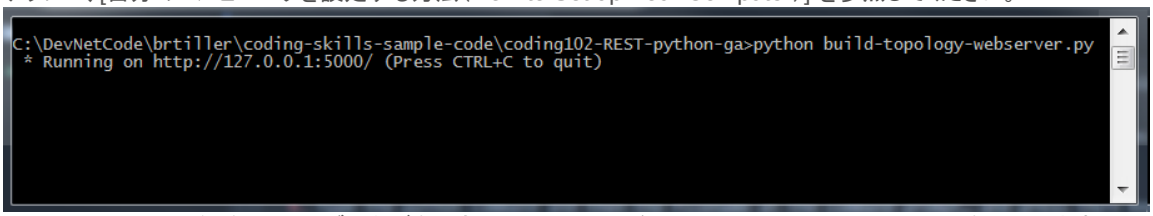
では、このコードの動作を見てみましょう。ここでは、主要なコードの変更に重点を置きます。

- *from flask import Flask*
  - flask python モジュールから、Flask オブジェクトをインポートします。
- *from flask import render_template, jsonify*
  - flask python モジュールから、render_template と jsonify という 2 つの関数をインポートします。
- *app = Flask(**name**)*
  - Flask Web アプリケーションをインスタンス化します。
- *@app.route("/")*
  - この下にある関数「def index()」が Flask Web アプリケーション変数「app」のデフォルトの Web ページとしてコールされるように指定します。
- *def index():*
  - インデックスで topology.html をロードする Flask 関数 render_template をコールします。
- *@app.route("/api/topology")*
  - この下にある関数「def topology()」が NeXt UI からコールされます。Flask の Web アプリケーション変数「app」の Web ページとして識別されます。
- *def topology():*
  - ネットワーク トポロジ データを JSON 形式で返します。
- *if **name** == "**main**":*
  - オプションのコードであり、インポートして関数を呼び出すのではなく、このモジュールを実行する場合に、このポイントの下のスクリプトを開始するということを明示するものです。一般的に、Python はすでにこの情報を持っていますが、このコードによって明確になります。
- *app.run()*
  - Flask Web アプリケーションを起動します。

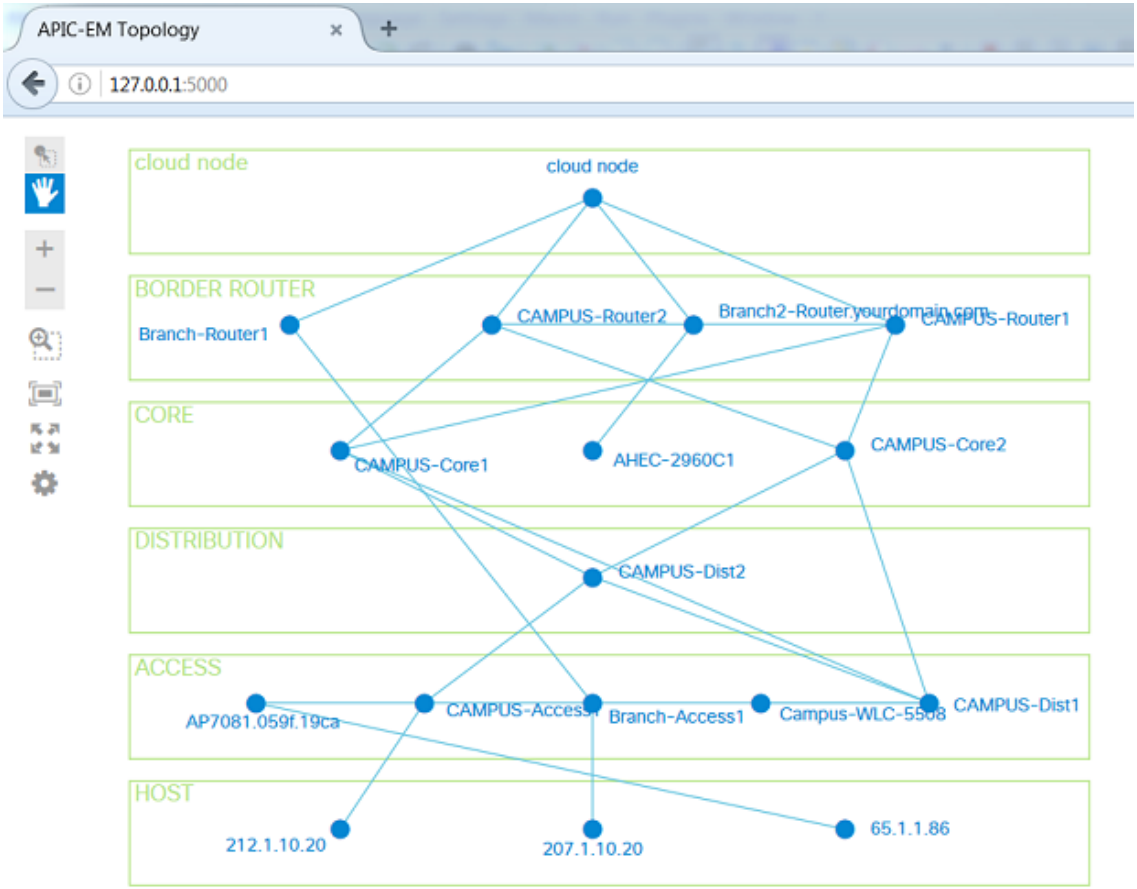
このサンプル コードを実行するには、次のようにします。

1. **coding102-REST-python-ga** ディレクトリに移動します。ターミナルで **cd \DevNetCode\<your-name>\coding-skills-sample-code\coding102-REST-python-ga** と入力します。
2. **controller** 変数に、APIC-EM コントローラの IP または DNS で引ける名前を割り当てます。
   - **build-topology-web-server.py** ファイルを開きます。たとえば、Windows では **notepad build-topology-web-server.py** と入力します。
   - *自分の APIC-EM コントローラを使用していない場合は、*DevNet サンドボックスの Always-On APIC-EM ラボ（https://sandboxapic.cisco.com）を使用します。
     - controller='sandboxapic.cisco.com'
3. ファイルを保存します。エンコーディング タイプがオプションの場合は、**UTF-8** を選択します。
4. Python コマンドを入力後、コマンド プロンプトにファイル名を入力して、リターン キーを押します。
   - *Windows では、***py -3 build-topology-web-server.py** と入力するか、**python build-topology-web-server.py** と入力します。
   - *Mac OS または Linux の場合は、***python3 build-topology-web-server.py** と入力します。

5. Flask Web サーバは、IP 127.0.0.1、ポート 5000 でリスニングを開始します。もし、*「インポート エラー:「flask」という名前のモジュールはありません(ImportError: No module named 'flask')」*というエラーが表示された場合は、ステップ 1、[自分のコンピュータを設定する方法(How to Set up Your Computer)] を参照してください。



6. Chrome や Safari などの Web ブラウザを開き、URL フィールドに URL **http://127.0.0.1:5000** を入力します。次のような結果が表示されます。



確認事項

- ステップ 5 で描画したトポロジと、このトポロジの最初の 3 つの階層を比べてみます。それらは一致しますか。
- テンプレート ディレクトリに移動して、topology.html ファイルを開きます。< *script type=text/javascript* > というブロック内にある javascript ブロックで、幅と高さを 800 から 400 に変更します。プログラムを再度実行します。何が変わりましたか。

おめでとうございます。コーディング 102 はこれで完了です。