SDWAN Programmability Lab Guide vE

July 11, 2019

Jed Demeule

Table of Contents

Lab Overview

Lab Flow

Lab 1: Exploring the REST API
Lab 2: Using Postman with the REST API
Lab 3: Installing Python and Tools
Lab 4: Using Python and the REST API to Script
Lab 5: Using the REST API to Gather Information
Lab 6: Using the REST API to Monitor the deployment
Lab 7: Using the REST API to Configure SD-WAN devices

Accessing the lab

In this task, you will access the NterOne Lab from your computer using the credentials supplied by your instructor.

Enter the credentials and URL in the tables below:

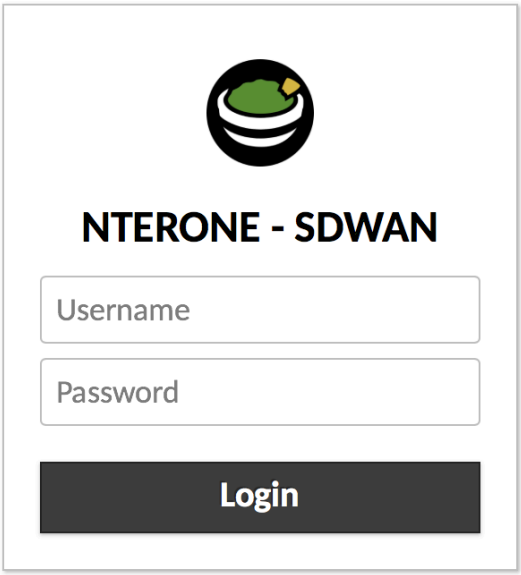
Username:	Password:
sdwan-XX (where XX is your assigned 2 digit pod number)	Instructor provided

Access URL:
http://

Activity Procedure

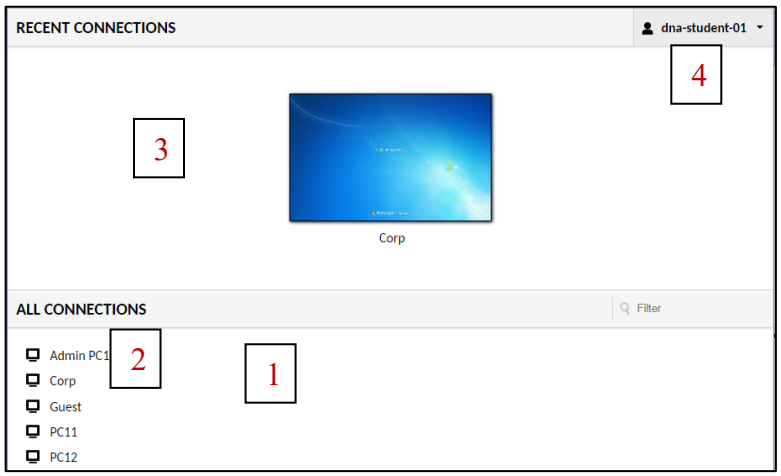
Complete these steps:

Step 1 Use a web browser of your choice, and browse to the **http://labs.nocular.com**. You should see the login screen below:



Step 2 Enter the credentials provided by your instructor and click **Login.**

Step 3 Observe the list of connections available **(1),** the type of devices **(2)**, the Recent Connections pane **(3),** and the Menu Bar **(4).**



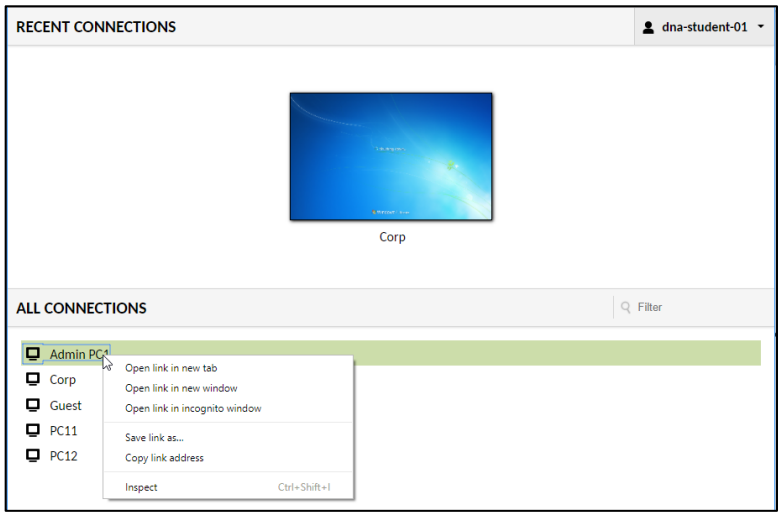
Step 4 Each device in your pod is listed, alphabetically, under "ALL CONNECTIONS" in the lower half of the screen

Step 5 A connection with a monitor icon provides a desktop, either through RDP or VNC; a connection with the prompt icon (>_) provides a **Telnet or SSH console**

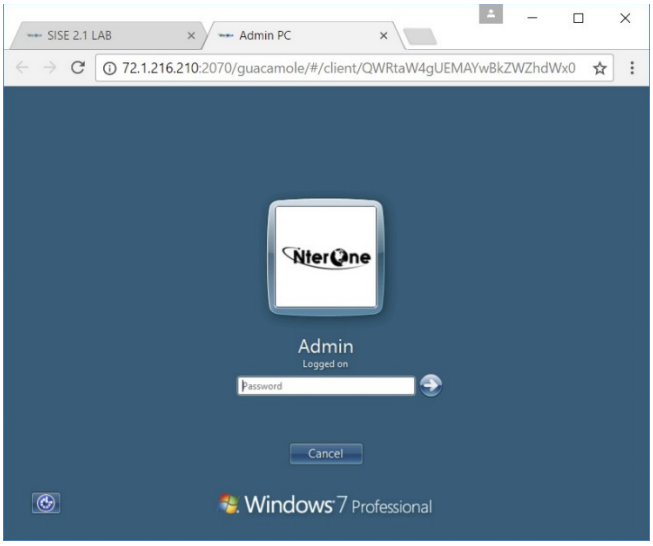
Step 6 For convenience recently accessed devices appear under "RECENT CONNECTIONS" at the top of the screen

Step 7 The menu in the upper left shows the username you are logged in under. Click the downwards arrow to open a dropdown menu, to access settings and to logout

Step 8 Try opening a session**. Right-click** on a server or PC (monitor icon), and select **open link in new tab**. Alternatively, hold down **COMMAND/CONTROL** and **left-click** the device to open the connection in a new tab.



Step 9 The connection opens in a new tab. Login to the device. The Admin-PC's Admin account password is **admin**.



Step 10 **Use the onboard keyboard to provide access to the CTRL-ALT-DEL key sequence.**

To access the on-board keyboard press CTRL-ALT-SHIFT

Activity Verification

You have completed this task when you obtain the following results:

- Successfully logged into the NterOne Lab environment
- Accessed a device thru your browser (uses RDP)

Topology – All Labs

In the Lab topology for the labs.

1 Admin PC per pod

1 vManage per pod
https://1.4.28.28

Lab 1 – Setting up the Admin PC for Development

To access the vManage GUI and REST API use the http addresses listed below.

Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

Step 1. Navigate to http://git-scm.com/downloads.
Step 2. Notice that there are versions for Windows and Linux.
Step 3. Since we are on Ubuntu you can simply install git by entering from the terminal **sudo apt-get install git** when prompted for a password enter **ubuntu**
Step 4. From within the terminal, enter the command **`git --version`**. You should get output indicating the version of git installed.

```
ubuntu@zzDNA:~$ sudo apt-get install git
sudo: unable to resolve host zzDNA: Connection timed out
[sudo] password for ubuntu:
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.7.4-0ubuntu1.6).
The following packages were automatically installed and are no longer required:
  libllvm5.0 linux-headers-4.13.0-41 linux-headers-4.13.0-41-generic
  linux-image-4.13.0-41-generic linux-image-extra-4.13.0-41-generic
  snapd-login-service
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 57 not upgraded.
ubuntu@zzDNA:~$ git --version
git version 2.7.4
ubuntu@zzDNA:~$
```

Step 5. You should be in your home directory, you will download the class files using Git

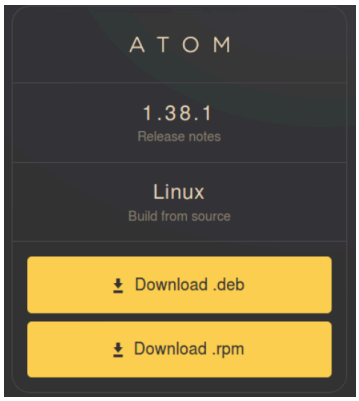
Step 6. Attempt to clone a repository from GitHub by entering **git clone https://github.com/deplorablejed/SDWAN-Programmability**.

```
ubuntu@zzDNA:~$ git clone https://github.com/deplorablejed/SDWAN-Programmability
Cloning into 'SDWAN-Programmability'...
remote: Enumerating objects: 66, done.
remote: Counting objects: 100% (66/66), done.
remote: Compressing objects: 100% (64/64), done.
remote: Total 66 (delta 26), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (66/66), done.
Checking connectivity... done.
```

Step 7. View the contents of what was just downloaded by entering the commad **`ls SDWAN-Programmability | more.`** These files will be used in later labs.

```
ubuntu@zzDNA:~$ ls SDWAN-Programmability | more
FTPPolicy.txt
package_config.ini
PostInventoryToWebex.py
PythonLab5Final.py
PythonLab6Challenge.py
PythonLab6Final.py
PythonLab6Source.txt
PythonLab7Final.py
PythonLab7.py
PythonLab7Source1.txt
PythonLab7Source2.txt
PythonLab8Final.py
PythonLab8Source.py
PythonTest.py
README.md
requirements.txt
SD-WAN Postman Collection.json
SD-WAN Postman environment.json
sdwan.py
```

Step 8. Go to http://atom.io and download the Atom text editor which will be used in class using the .deb file.



Step 9. Install Atom and when prompted enter a password of **ubuntu**.

Step 10. If you do not already have a Webex Teams account, set a free account up now on your personal computer, NOT the ubuntu desktop used in this class.

NOTE: Webex teams will be used in the labs and also by the instructor to distribute information and files.

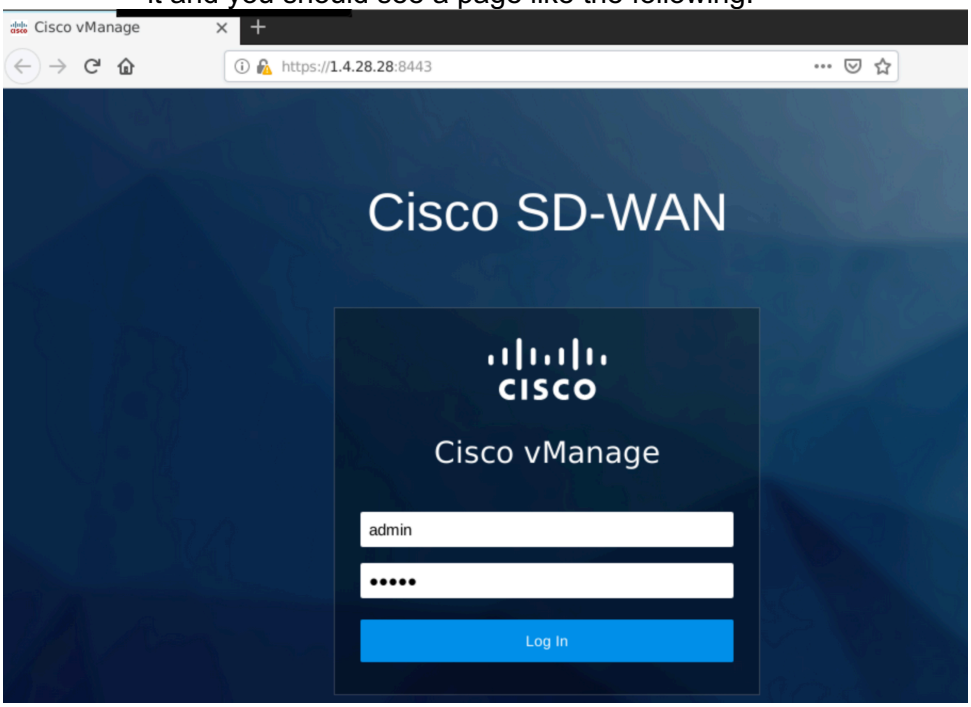
Step 11. Provide your instructor with your email address associated with the account and ask him or her to add you to the classroom space.

Lab 2 - Exploring the REST API

To access the vManage GUI and REST API use the http addresses listed below.

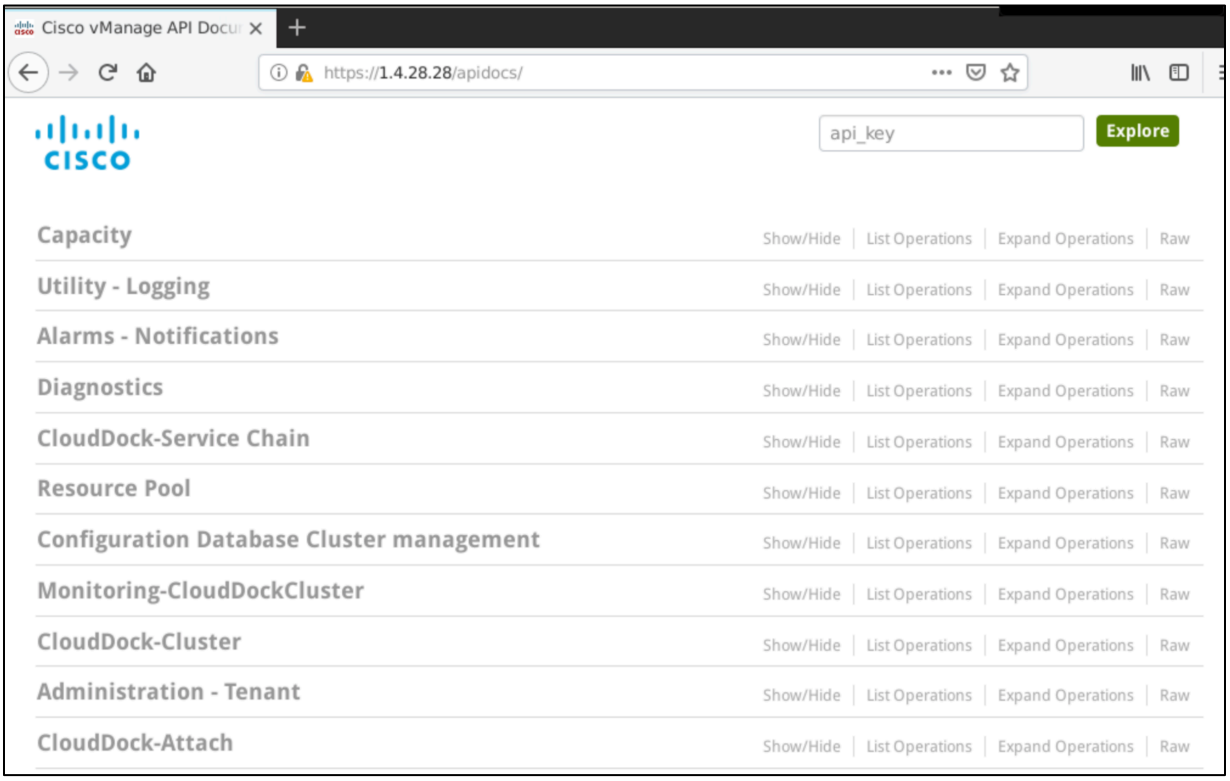
Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

Step 1. From the remote desktop of your admin PC open the firefox web browser.
Step 2. Enter a URL of **https://1.4.28.28**. When presented with a security warning bypass it and you should see a page like the following.

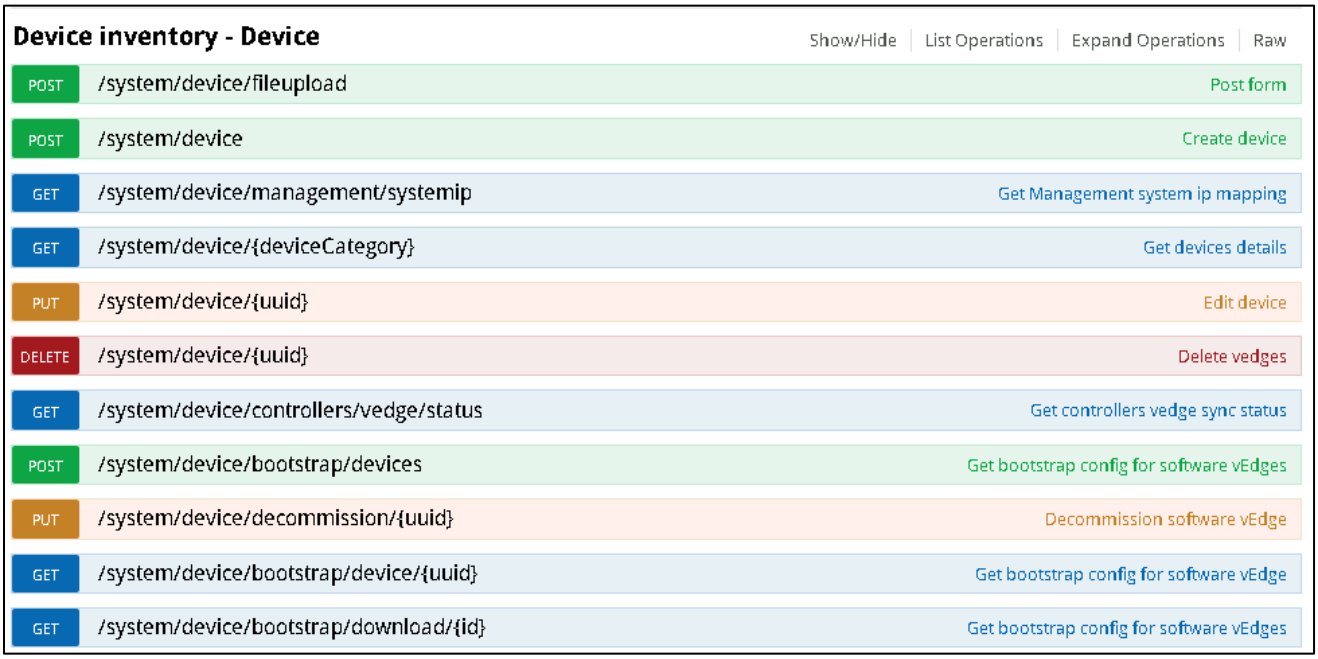


Step 3. Enter a username of **`admin`** and a password of **`admin`**

Step 4. Now change the URL by going to https://1.4.28.28/apidocs.



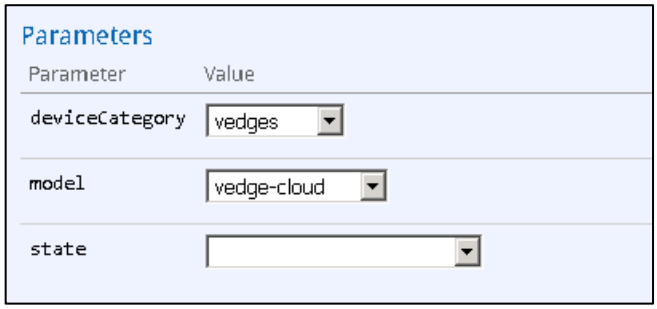
Step 5. To the right of **Device inventory – Device** select the **Show/Hide** link and notice all of the POST and GET actions available.



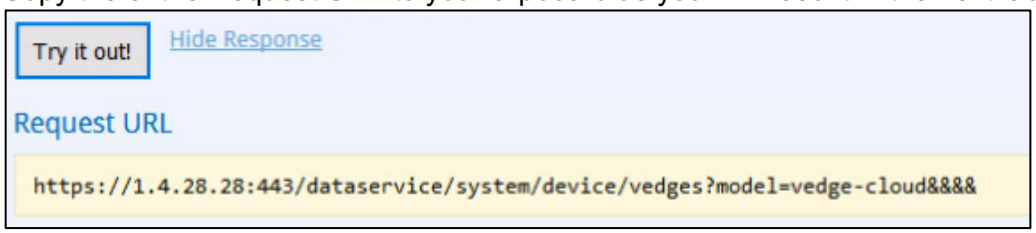
Step 6. Select **Get device details** on the **GET /system/device/{deviceCategory}** action. From the drop downs select the following:

- deviceCategory: **vedges**

- model: **`vedge-cloud`**
- state: ***`blank`***



Step 7. At the bottom of the section press the **Try it out!** button.
Step 8. Notice the **Request URL** contains the 2 parameters we set above.
Step 9. Copy the entire Request URL to your clipboard as you will need it in the next lab.



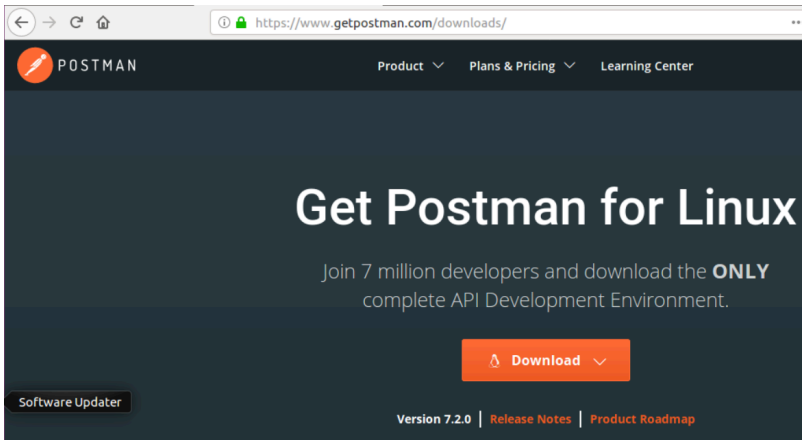
Step 10. Look in the response body which has quite a bit of output. Scroll thru the output and notice there are eight vEdge routers configured in vManage.

Lab 3 - Using Postman with the REST API

Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

In this lab you will use a tool by Google called Postman to test REST API calls. This tool is useful to test you syntax and develop applications or scripts that use the REST API.

Step 1. Open up a URL to **https://www.getpostman.com/downloads** and select the **Download** button and select the 64 bit version.



Step 2. You will be prompted to open the downloaded file with Archive Manager press **OK.**

Step 3. Press the **Extract** button and leave at the default location which is Home

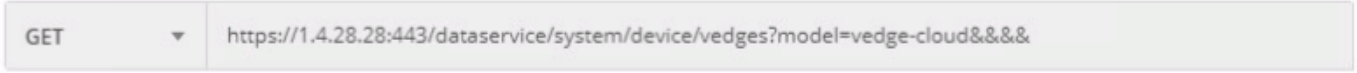
Step 4. When the extraction is done press the **Show the Files** button and open up the Postman folder.

Step 5. Start the application by selecting the Postman icon.

Step 6. Select at the bottom of the page in small print "**Skip signing in and take me straight to the app**".

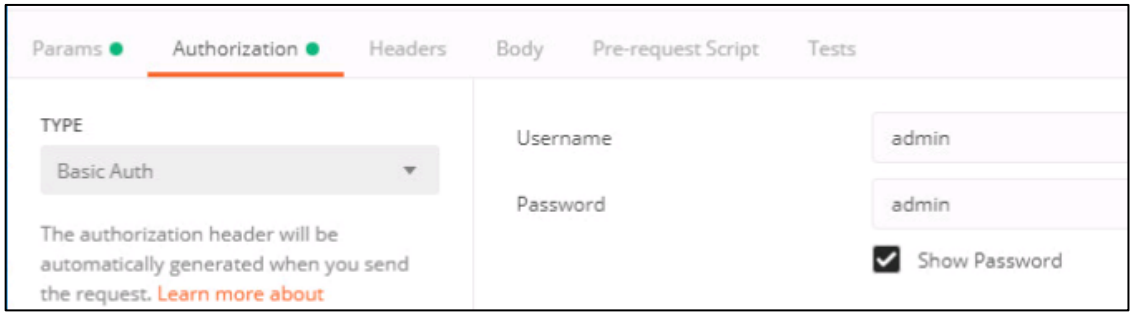
Step 7. Press the New button in orange in the top left, enter a name of **My first Postman**, Create a new collection and give the collection a name of **SDWAN**.

Step 8. Create a collection of and fill out the URL for the GET by using the https://1.4.28.28/apidocs page and pasting the URL or manually typing it.



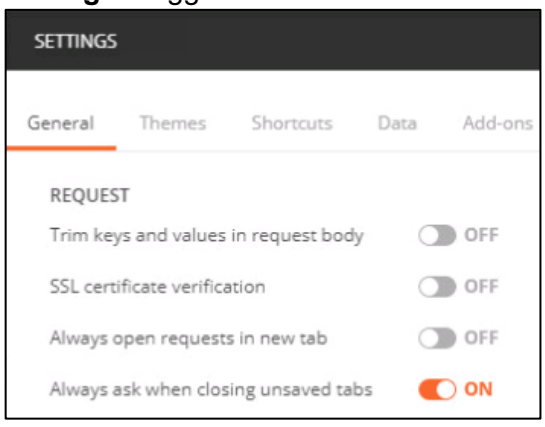
Step 9. Notice the **Params** tab is populated with `model` and `vedge-cloud`

Step 10. Next select the **Authorization** tab and select the type to **`Basic Auth`** and then enter a username of **`admin`** with a password of **`admin`**.



Step 11. Press the blue **Send** button and notice there is an error.

Step 12. Allow the use of self signed certificates, which in the case of our lab is what we are using. Click the **wrench** icon in the upper right of the interface and select **Settings**. Toggle the **SSL certificate verification** to **`OFF.`**

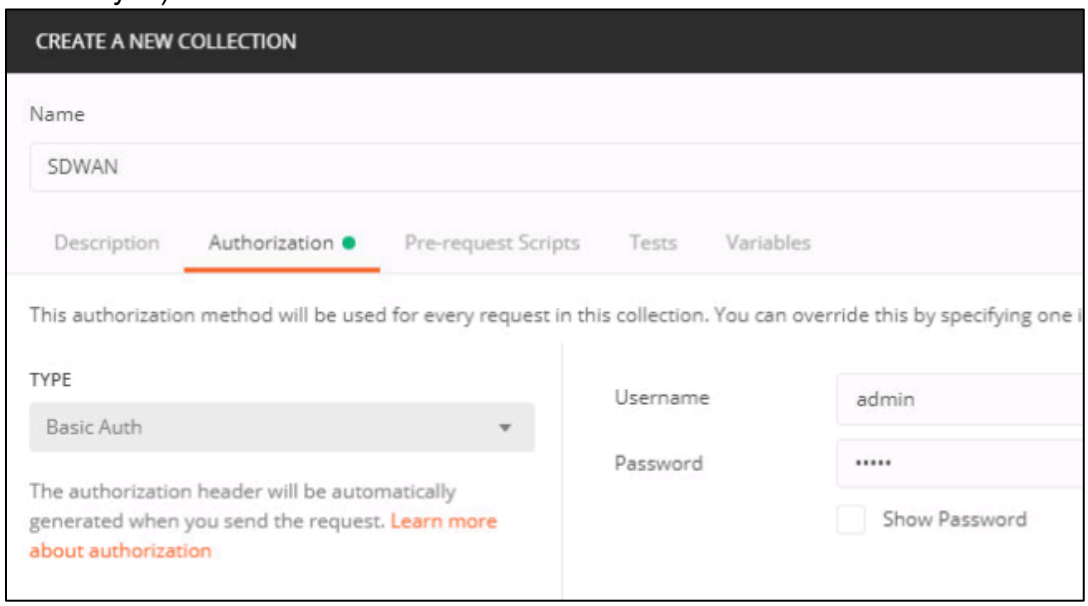


Step 13. Close the Settings window and try to send the GET request again. This time it should be successful.

Step 14. Starting at line 207-252 take a couple minutes to look at all the returned information about the vEdge with a hostname of Branch-vE2


```
207         "deviceType": "vedge",
208         "serialNumber": "8f294336",
209         "ncsDeviceName": "vedge-06f4b448-5879-43bb-bc28-3b6828fb5f2a",
210         "configStatusMessage": "In Sync",
211         "templateApplyLog": [
212             "[5-Feb-2019 15:50:57 UTC] Configuring device with cli template: Branch-vE2",
213             "[5-Feb-2019 15:50:57 UTC] Generating configuration from template",
214             "[5-Feb-2019 15:50:57 UTC] Checking and creating device in vManage",
215             "[5-Feb-2019 15:51:20 UTC] Device is online",
216             "[5-Feb-2019 15:51:20 UTC] Updating device configuration in vManage",
217             "[5-Feb-2019 15:51:21 UTC] Pushing configuration to device",
218             "[5-Feb-2019 15:54:29 UTC] Template successfully attached to device"
219         ],
220         "uuid": "06f4b448-5879-43bb-bc28-3b6828fb5f2a",
221         "managementSystemIP": "0.0.0.0",
222         "templateStatus": "Success",
223         "chasisNumber": "06f4b448-5879-43bb-bc28-3b6828fb5f2a",
224         "configStatusMessageDetails": "",
225         "configOperationMode": "vmanage",
226         "deviceModel": "vedge-cloud",
227         "deviceState": "READY",
228         "validity": "valid",
229         "platformFamily": "vedge-x86",
230         "vedgeCertificateState": "certinstalled",
231         "vedgeCSR": "-----BEGIN CERTIFICATE REQUEST
```


Step 15. Set the Authorization to **`Basic Auth`** (the username and password should be auto filled for you). Press the **Save** button when done.



Step 16. You should use Postman as you are developing applications and script to test your API syntax and results.

Step 17. End of lab

Lab 4 – Installing Python and Setting up your Python Development Environment

To access the vManage GUI and REST API use the http addresses listed below.

Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

Step 1. From the remote desktop of your admin PC open the folder in your home directory called **SDWAN-Programmability.**

Step 2. Open the file called **Install Python 3.7.txt** in a text editor. These are the commands that were used to install Python 3.7.3 on the Ubuntu desktop. Python 3.7.3 is already installed and ready on the Ubuntu. The one big line to paste in is shown below (if prompted for a password enter **ubuntu**):

```
sudo apt-get install build-essential checkinstall -y && sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev libffi-dev zlib1g-dev -y && cd /usr/src && sudo wget https://www.python.org/ftp/python/3.7.3/Python-3.7.3.tgz && sudo tar xzf Python-3.7.3.tgz && cd Python-3.7.3 && sudo ./configure --enable-optimizations -y && sudo make altinstall
```

Next you are going to create Python Virtual Environments. Python Virtual Environments are a method of creating isolated "environments" where specific versions of Python can be installed along with independent sets of libraries and dependencies.
Virtual Environment usage is a very common, and recommended practice when working in Python.

Step 3. First we need to intall the venv module, do this by running the command **sudo apt-get install python3-venv**, when prompted for a password enter **ubuntu** and enter **y** to allow the install.

Step 4. Make sure you are in your home directory (cd /home/ubuntu). Failure to do so will result in permission errors when you try and run the following commands.

Step 5. First create a Python 3.7 virtual environment using the **`venv`** module included with Python 3. The virtual environment can be any thing that you want, but in our lab lets call it **hello**. Enter the command **python3.7 -m venv hello**.

```
ubuntu@zzDNA:~$ python3.7 -m venv hello
```

Step 6. Now "activate" the environment. Look for the name of the virtual environment to be enclosed in parenthesis after activation. Enter the command **source hello/bin/activate**.

```
ubuntu@zzDNA:~$ source hello/bin/activate
(hello) ubuntu@zzDNA:~$
```

Step 7. Now in the virtual environment enter the command **python3.7 -V** and verify Python 3.7.3 is active in this virtual environment.

```
(hello) ubuntu@zzDNA:~$ python3.7 -V
Python 3.7.3
(hello) ubuntu@zzDNA:~$
```

Step 8. To tear down and destroy the virtual environment enter the command **deactivate**.

Virtual environments are an elegant way to uncluttered your global Python environment, and isolate projects from each other. At first, they might sound somewhat mysterious, but they are conceptually simple.

When you create a virtual environment (like you did with the **`python3.7 -m venv <virtual environment name>`** command), the virtual environment module **`venv`** creates a new folder using the name you provided. Looking inside that folder, you will see something like the following:

```
venv/
├── bin
├── include
├── lib
└── pip-selfcheck.json
```

Now, when working inside an activated virtual environment and you **`pip install`** some *library*, where do you think that gets installed?
That's right! ...in the **`/lib`** directory.
What about if you **`pip install`** some *executable* script?
Also correct! ...in the **`/bin`** directory.

You see, a virtual environment is essentially redirecting where packages get installed and where Python looks to find them. Instead of installing packages in the global environment, they are instead installed in the virtual environment directory that you told `venv` to create. You can create these virtual environments in the same directory as your project files (though you usually exclude them from version control), and then everything stays nice and neat. When you go to work on a project, you can simply activate its virtual environment and you have all the packages you need to work on that project. When you want to switch from working on Project A to working on Project B, you deactivate Project A's virtual environment and then activate Project B's.

When you are done with a Project and perhaps delete it from your workstation, the virtual environment directory and the packages installed in are also deleted. No more cluttering up directories with unneeded files.

Step 9. You will need to know how to start a virtual environment throughout this course as your labs will be run in a virtual environment.

Step 10. Start a virtual environment called **sdwan** based on the previous steps (you may even want to make a shell script to automate this)

Step 11. Open a terminal run the command **`pip install requests`**. This module will be used in the next task.

```
ubuntu@zzDNA:~$ source hello/bin/activate
(hello) ubuntu@zzDNA:~$ pip install requests
Collecting requests
  Downloading https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a00aba99ae0f828be89
d72b2190f27c11d4b7fb/requests-2.22.0-py2.py3-none-any.whl (57kB)
    100% |████████████████████████████████| 61kB 2.1MB/s
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 (from requests)
  Downloading https://files.pythonhosted.org/packages/e6/60/247f23a7121ae632d62811ba7f273d0e58972d75
e58a94d329d51550a47d/urllib3-1.25.3-py2.py3-none-any.whl (150kB)
    100% |████████████████████████████████| 153kB 4.2MB/s
Collecting certifi>=2017.4.17 (from requests)
  Downloading https://files.pythonhosted.org/packages/69/1b/b853c7a9d4f6a6d00749e94eb6f3a041e342a885
b87340b79c1ef73e3a78/certifi-2019.6.16-py2.py3-none-any.whl (157kB)
    100% |████████████████████████████████| 163kB 7.9MB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Using cached https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb1ddec7ca55ec75
10b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl
Collecting idna<2.9,>=2.5 (from requests)
  Downloading https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd03869a46fe7226e
7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl (58kB)
    100% |████████████████████████████████| 61kB 10.4MB/s
Installing collected packages: urllib3, certifi, chardet, idna, requests
Successfully installed certifi-2019.6.16 chardet-3.0.4 idna-2.8 requests-2.22.0 urllib3-1.25.3
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Step 12. Run the command **`pip install configparser`**. This module will be required in the next task.

```
(hello) ubuntu@zzDNA:~$ pip install configparser
Collecting configparser
  Downloading https://files.pythonhosted.org/packages/ba/05/6c96328e92e625fc31445d24d75a2c92ef9ba34f
c5b037fe69693c362a0d/configparser-3.7.4-py2.py3-none-any.whl
Installing collected packages: configparser
Successfully installed configparser-3.7.4
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Step 13. Enter the command **python3.7** to enter interactive mode, notice the version of python that you installed and that the prompt is now a >>>

```
ubuntu@zzDNA:~$ python3.7
Python 3.7.3 (default, Jun 16 2019, 16:02:56)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

NOTE: You will manually install modules in this lab, later in the course you will learn how to automate this with a requirements.txt file

Step 14. Lets make sure out modules can be loaded correctly by entering the following 3 commands following each with a Carriage Return. Ensure that no errors are generated.

```
import requests
import urllib3
import configparser
```

```
(hello) ubuntu@zzDNA:~$ python3.7
Python 3.7.3 (default, Jul 10 2019, 23:06:11)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> import urllib3
>>> import configparser
>>>
```

Step 15. Enter the following in interactive mode, Python program print command **print ("Hello World!")** and press **<Enter>** and notice "Hello World!" is output.

```
>>>print ("Hello World!")
Hello World!
>>>
```

At any point in time, to exit the Python interactive mode prompt, press **<CTRL>** + **d** or type **exit()** and then **<Enter>**.

Step 16. Enter From the command prompt session, configure two Python variables "a" and "b". Assign the value to each as follows:

- Variable a value: "SDWAN"
- Variable b value: 1234

Note A single variable is used to represent a piece of data or a set of data.

From the Terminal session in the interactive Python mode on the PC2 enter the **a="SDWAN"** command and press **<Enter>**. Then, print the value of variable **a** using the **print(a)** command. Press **<Enter>** again to see the result.

Configure using the value of variable **b** by entering the command **b=1234 <Enter>**. Then print the value of variable b using the command **print(b) <Enter>**. The output should look similar to this:

```
>>> a="SDWAN"
>>> print(a)
SDWAN
>>> b=1234
>>> print(b)
1234
```

Note This output shows that a variable can be assigned to a text value (delimited by single or double quotes) or a numerical value (with no quotes). In the examples, you use the print statement to show the contents of a variable. In the Python interpreter, you can examine the contents of a variable by simply typing the variable name.

Step 17. From the command prompt session, configure a new Python list variable **c.** Then, define the following list items values for index values 0, 1, and 2:

- c[0] = "Cisco"
- c[1] = 'SDWAN'

- c[2] =4321

In the command prompt session, with the Python interpreter in the interactive mode, enter the **c =["Cisco", 'SDWAN',4321] <Enter>**. This creates a new list with **Cisco** in index 0 first, then **SDWAN** in index 1, and then **4321** in index 2. In order to output each value of the list **c**, enter the **c[0] <Enter>**, then **c[1] <Enter>**, and finally **c[2] <Enter>**. You should see the following result:

```
>>> c=["Cisco",'SDWAN',4321] <Enter>
>>> c[0] <Enter>
'Cisco'
>>> c[1] <Enter>
'SDWAN'
>>> c[2] <Enter>
4321
```

Step 18. From the command prompt session, configure Python **for** loop statement that will print the 3 indexes that you defined in the list called c.

Enter the **for item in c: <Enter> <Tab>print(item) <Enter>** command and then **<Enter>** again after the three dots in interpretive mode as follows to output the content of the **c** list. Your output should be similar to this:

```
>>> for item in c:<Enter>
…    <tab>print(item)<Enter>
…<Enter>
Cisco
SDWAN
4321
>>>
```

Step 19. From the command prompt session, configure and use a Python for loop statement to print a range of values from 1 to 3.

If you are familiar with for loop statements from another language, they may look a little different in Python. You probably remember that loops used for iterating through a series of numbers. We can do this in Python as well, but we need to use a special syntax.

Python **for** loop operator iterates some integer variable in increasing or decreasing order. Such a sequence of integer can be created using the function **range(min_value, max_value)**.

The Python **range(min_value, max_value)** function generates a list of numbers, which is generally used to iterate over with **for** loops. Function **range(min_value, max_value)** generates a sequence with numbers **min_value**, **min_value + 1**, **…**, **max_value – 1**. The last number is not included.

Enter the following **for** loop statement **for n in range(1,4):<Enter><tab>print (n)<Enter>** and then again **<Enter>** after the 3 dots in the interactive mode and notice that the output of this statement are printed integer values 1, 2, and 3.

The result may surprise you. Python stops before it gets to 4. This is normal, and you have to take this into account when using the for loop statement range. In most cases this means the range max value should be set to n+1 of you want to iterate through the loop n times.

Your output should be similar to this:

```
>>> for n in range(1,4):<Enter>
...     <tab>print(n)<Enter>
...<Enter>
1
2
3
```

Step 20. From the Admin PC session, you will create and store data in a dictionary data structure called switch, which is similar to a list data structure. This dictionary will store the following paramters:

- vEdge: **Branch-vE1**
- IP address: **10.80.88.111**
- Username: **admin**
- Password: **admin**

Unlike a list, however, you can retrieve information from dictionaries using keys. Instead of a number we can access specific elements of a dictionary with key names. We still enclose the key in brackets, as we did with lists.

Enter the following in the Python interpreter interactive mode:

```
>>>vEdge={'name':'Branch-vE1','ip':'10.80.88.111','username':'admin','password':'admin'}<Enter>
>>>vEdge['name'] <Enter>
'Branch-vE1'
>>>vEdge['password'] <Enter>
'admin'
>>>
```

Note If you are having trouble with the dictionary, be sure you used curly brackets to define it, that you enclosed each key and value with quotes, and that you put commas between each value. Also, note that while we define the dictionary with curly brackets, we access each element with square brackets.

Step 21. Quit the interpreter with <CTRL> + d
Step 22. Enter deactivate to tear down your virtual environment.
Step 23. End of lab

Lab 5 - Using Python Scripting and the REST API

In lab 5 you will construct a python script that opens an authentication session using the API of the vManage. The credentials that are used to authenticate are not defined in the python script but rather are in a .ini file that the python script parses. This allows the credentials to be shared by multiple python scripts and enables the credentials to be changed at a later time without changing the scripts.

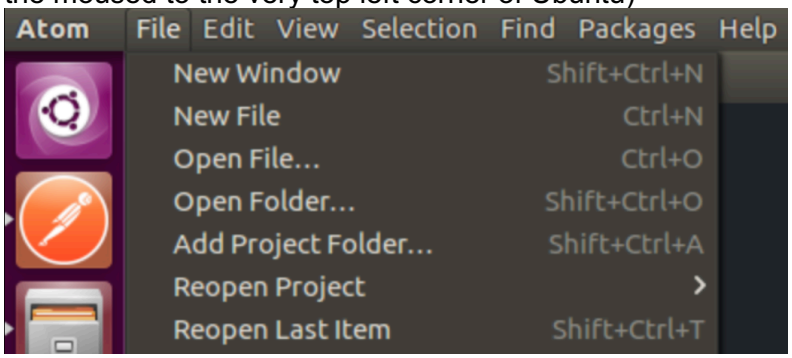
NOTE: The lab guide is in the repository that you downloaded with git. It is possible to copy and paste from the PDF although this can sometimes introduce errors. Word puts in smart quotes in place of quotes by default and these will not work in python code. This lab guide has no curly quotes so copy and paste should work better although not perfect.

To access the vManage GUI and REST API use the http addresses listed below.

Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

Step 1. From the remote desktop of your admin PC navigate to the start button and go to **Start → Atom**.

Step 2. In Atom use the **File → New File** menu to create a new python script (Hint: move the moused to the very top left corner of Ubuntu)



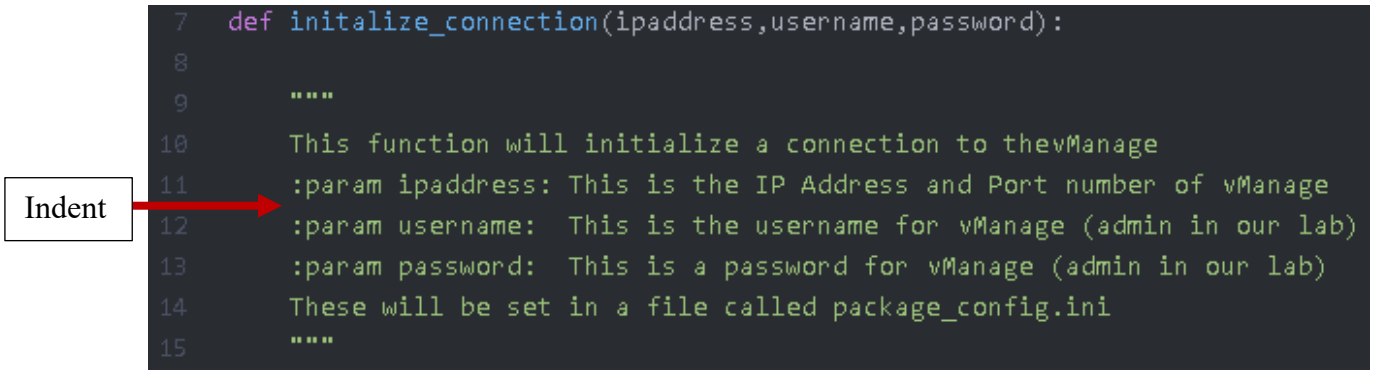
Step 3. In the new file you will add the following lines to gain access to the code in additional modules:

```
import requests
import urllib3
import configparser
```

```
import requests
import urllib3
import configparser
```

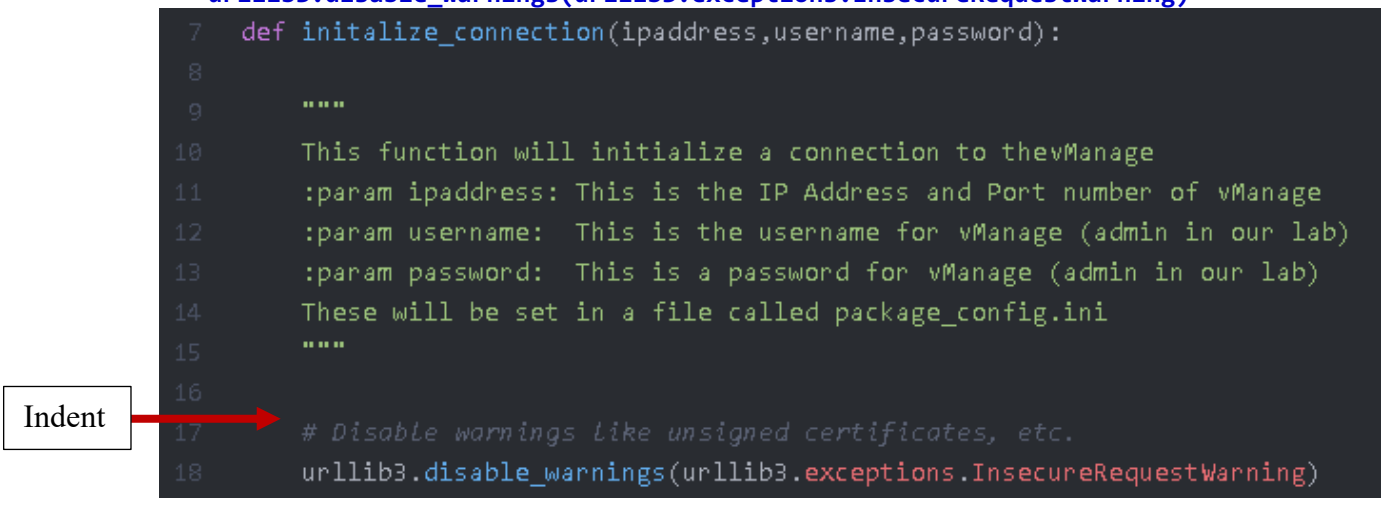
Step 4. Lets start making a connection to the vManage, you will start by entering the following lines into the Atom editor (tabs are important in python):

```
def initialize_connection(ipaddress,username,password):
    """
    This will initialize the connection to the vManage.
    :param ipaddress: this the IP address and port of vManage
    :param username: This is the username for vManage (admin in our lab)
    :param password: The password for vManage (admin in our lab)
    These will be set in a file called package_config.ini
    """
```



Step 5. Next we will add to the bottom of the script the following lines indented below the previous step.

```
    # Disable warnings like unsigned certificates, etc.
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

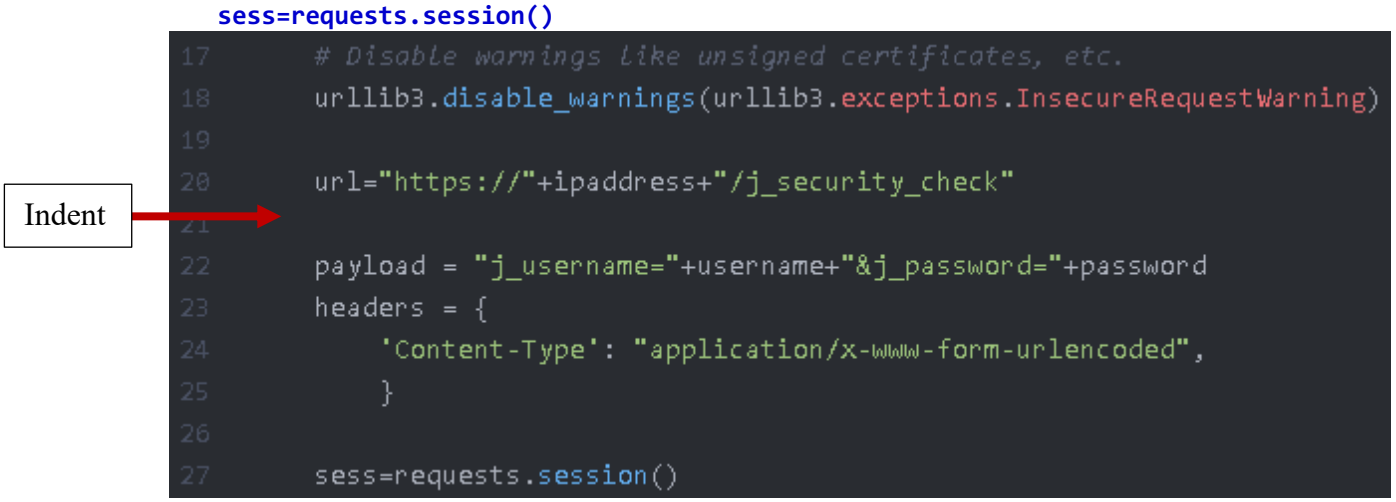


Step 6. Next you will define where to authenticate and how to authenticate to the vManage. Add the following lines at the same indent level as the previous step:

```
url="https://"+ipaddress+"/j_security_check"

payload = "j_username="+username+"&j_password="+password
headers = {
    'Content-Type': "application/x-www-form-urlencoded",
    }

sess=requests.session()
```



Step 7. Next you will define how to handle what to do if an error is received when connecting to vManage. Enter the following lines at the same indent level as the previous step:

```
# Handle exceptions if we cannot connect to the vManage
try:
    response = sess.request("POST", url, data=payload,headers=headers,verify=False,timeout=10)
except requests.exceptions.ConnectionError:
    print ("Unable to Connect to "+ipaddress)
    return False
return sess
```

Step 8. Next you will read the **package_config.ini** file and populate the local variables with date found within. Enter the following lines starting with **no** indentation:

```
# Open up the configuration file and get all application defaults
try:
    config = configparser.ConfigParser()
    config.read('package_config.ini')

    serveraddress = config.get("application","serveraddress")
    username = config.get("application","username")
```

```
    password = config.get("application","password")
    systemip = config.get("application","systemip")
except configparser.Error:
    print ("Cannot Parse package_config.ini")
    exit(-1)

print ("Viptela Configuration:")
print ("vManage Server Address: "+serveraddress)
print ("vManage Username: "+username)
```

```
39  # Open up the configuration file and get all application defaults
40  try:
41      config = configparser.ConfigParser()
42      config.read('package_config.ini')
43
44      serveraddress = config.get("application","serveraddress")
45      username = config.get("application","username")
46      password = config.get("application","password")
47      systemip = config.get("application","systemip")
48  except configparser.Error:
49      print ("Cannot Parse package_config.ini")
50      exit(-1)
51
52  print ("Viptela Configuration:")
53  print ("vManage Server Address: "+serveraddress)
54  print ("vManage Username: "+username)
```

Step 9. Finally lets initialize the connection to vManage, enter the following lines starting with **no** indentation:

```
session= initialize_connection(serveraddress,username,password)
if session != False:
   print ("Successful you will issue API commands here in later labs")
```

```
56  session=initalize_connection(serveraddress,username,password)
57  if session != False:
58      print ("Successful you will issue API commands here in later labs")
```

Step 10. Save your script as **PythonLab5.py**

Step 11. Go to the Terminal and change to the **cd SDWAN-Programmability** directory (if not already there)

Step 12. Next create a Python 3 virtual environment using the **venv** module included with Python 3. Enter the command **python3.7 -m venv sdwan**.

Step 13. Now "activate" the environment. Look for the name of the virtual environment to be enclosed in parenthesis after activation. Enter the command **source sdwan/bin/activate**.

Step 14. Leave this virtual environment activated for the rest of the labs and if you closed it repeat the steps above to activate it again.

Step 15. Enter the command **pip install requests**

Step 16. Enter the command **pip install urllib3**

Step 17. Enter the command **pip install configparser**

Step 18. Enter **python3.7 PythonLab5.py** to execute the script you made.

```
(sdwan) ubuntu@zzDNA:~/SDWAN-Programmability$ python3.7 PythonLab5.py
Viptela Configuration:
vManage Server Address: 1.4.28.28
vManage Username: admin
Successful you will issue API commands here in later labs
```

Step 19. If you have errors you have 2 options, the first is to try and debug your script, the second is to go to the SDWAN folder on your desktop and use **File → Open** and open up the **PythonLab5Finished.py** script, save it as **PythonLab5.py** and run it.

Step 20. Do not proceed to any further labs until you have a functional script as this framework will be used in later labs.

Step 21. End of lab

Lab 6 – Using the vManage REST API to Gather Information

To access the vManage GUI and REST API use the http addresses listed below.

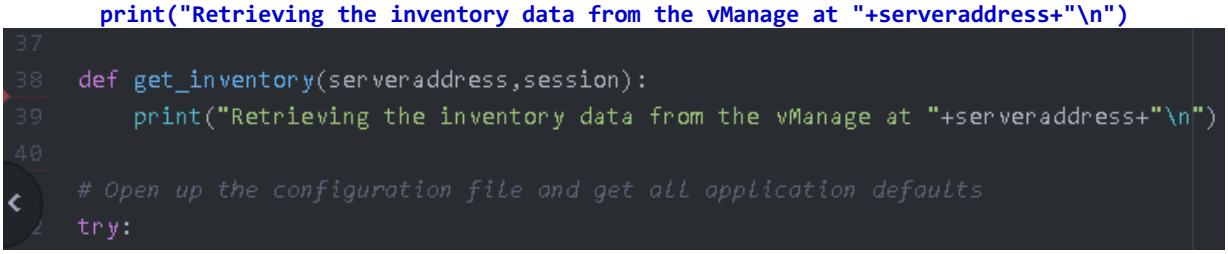
Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

Step 1. From the remote desktop of your admin PC navigate to the start button and go to **Start → Atom.**

Step 2. Open the finished script from lab 5 called **PythonLab5.py**. This will be the starting point for this lab.

Step 3. Before the section that starts with "**# Open up the configuration file and get all application defaults**" enter the following lines:

```
def get_inventory(serveraddress,session):
    print("Retrieving the inventory data from the vManage at "+serveraddress+"\n")
```



Step 4. Now you will define the REST API URL that will be used to fetch the inventory from a the vManage in our lab by entering the following below the previous step (ensure the indents match exactly the screenshot below when done):

```
    url = "https://" + serveraddress + "/dataservice/device"
    response = session.request("GET",url,verify=False,timeout=10)
    json_string = response.json()
    print(json_string)
```

Indent

Step 5. At the bottom of the script add an indented line

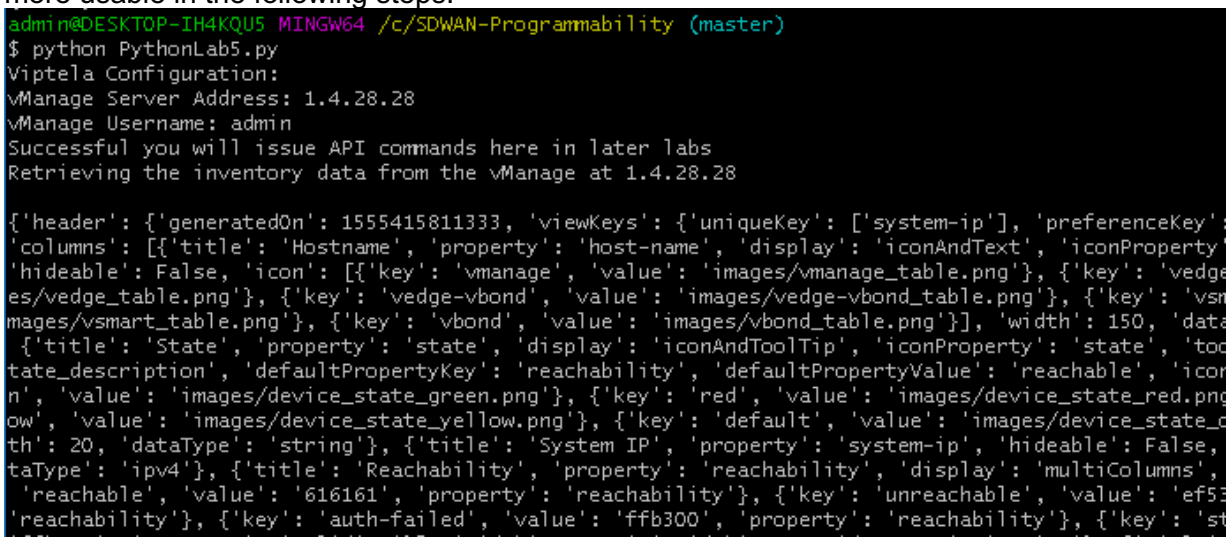
```
get_inventory(serveraddress,session)
```

```
session=initalize_connection(serveraddress,username,password)
if session != False:
    print ("Successful you will issue API commands here in later labs")
    get_inventory(serveraddress,session)
```

Step 6. Save the script as **PythonLab6.py**.

Step 7. Go to the Terminal and change to the SDWAN-Programmability directory with the command **cd SDWAN-Programmabiltiy** (if not already there) and execute the command **python PythonLab6.py**

Step 8. View the output and notice that all the devices in the SDWAN deployment are listed in the resulting json file returned from the http GET that was defined in the get_inventory definition. It is not easy to read the output lets see if we can make it more usable in the following steps.



Step 9. Comment out the **print (json_string**) by placing a # in front of it and add the following lines below.

```
for item in json_string['data']:
    print(item)
    print("=====================")
```

```
inv={}
for item in json_string['data']:
    print(item)
    print("=======================")
```

Step 10. Save the script as **PythonLab6.py** and then run the script again in the terminal, the results should look like the following:

```
admin@DESKTOP-IH4KQU5 MINGW64 /c/SDWAN-Programmability (master)
$ python PythonLab5.py
Viptela Configuration:
vManage Server Address: 1.4.28.28
vManage Username: admin
Successful you will issue API commands here in later labs
Retrieving the inventory data from the vManage at 1.4.28.28

{'deviceId': '172.1.0.18', 'system-ip': '172.1.0.18', 'host-name': 'vManage-1', 'reachability': 'reachable', 'status
': 'normal', 'personality': 'vmanage', 'device-type': 'vmanage', 'timezone': 'UTC', 'device-groups': ['"No groups"']
, 'lastupdated': 1555412615226, 'domain-id': '0', 'board-serial': '69DE12CE976BC070D5184152E0298516', 'certificate-v
alidity': 'Valid', 'max-controllers': '0', 'uuid': '618c3989-5dbe-4f92-95b8-cca81d1a466e', 'controlConnections': '11
', 'device-model': 'vmanage', 'version': '18.4.0', 'connectedVManages': ['"172.1.0.18"'], 'site-id': '1001', 'latitu
de': '37.666684', 'longitude': '-122.777023', 'isDeviceGeoData': False, 'platform': 'x86_64', 'uptime-date': 1555047
420000, 'statusOrder': 4, 'device-os': 'next', 'validity': 'valid', 'state': 'green', 'state_description': 'All daem
ons up', 'model_sku': 'None', 'local-system-ip': '172.1.0.18', 'total_cpu_count': '6', 'testbed_mode': False, 'layou
tLevel': 1}
======================
{'deviceId': '172.1.0.16', 'system-ip': '172.1.0.16', 'host-name': 'vSmart-1', 'reachability': 'reachable', 'status'
: 'normal', 'personality': 'vsmart', 'device-type': 'vsmart', 'timezone': 'UTC', 'device-groups': ['"No groups"'], '
lastupdated': 1555412683910, 'domain-id': '1', 'board-serial': '07F4E30F8E9368C48C066D1C5AB51980', 'certificate-vali
dity': 'Valid', 'uuid': '6e4d0357-e883-4931-853e-25e5a85ad979', 'controlConnections': '13', 'device-model': 'vsmart'
, 'version': '18.4.0', 'connectedVManages': ['"172.1.0.18"'], 'site-id': '1011', 'ompPeers': '7 (10)', 'latitude': '
37.666684', 'longitude': '-122.777023', 'isDeviceGeoData': False, 'platform': 'x86_64', 'uptime-date': 1555057920000
, 'statusOrder': 4, 'device-os': 'next', 'validity': 'valid', 'state': 'green', 'state_description': 'All daemons up
', 'model_sku': 'None', 'local-system-ip': '172.1.0.16', 'total_cpu_count': '2', 'testbed_mode': False, 'layoutLevel
': 2}
======================
{'deviceId': '172.1.0.17', 'system-ip': '172.1.0.17', 'host-name': 'vSmart-2', 'reachability': 'reachable', 'status'
```

Step 11. Next lets extract and print just some of the data items and display them. Lets extract the hostname, version, serial number and the number of control connections. Enter the following lines below the previous steps (use 3 spaces between the items in quotes).

```
for item in json_string['data']:
    print (item['host-name']+"   "+item['board-serial']+"   "+item['version']+"   "+item['controlConnections'])
```

```
inv={}
for item in json_string['data']:
   print(item)
   print("======================")
for item in json_string['data']:
   print (item['host-name']+"   "+item['board-serial']+"   "+item['version']+"   "+item['controlConnections']+"
return(inv)
```

Step 12. Save the script as **PythonLab6.py** and then run the script in the terminal, the results should look like the following (at the bottom of the output):

```
======================
{'deviceId': '172.1.30.4', 'system-ip': '172.1.30.4', 'host-name':
s': 'normal', 'personality': 'vedge', 'device-type': 'vedge', 'tim
lastupdated': 1555412632485, 'bfdSessionsUp': 12, 'domain-id': '1'
y': 'Valid', 'max-controllers': '0', 'uuid': '276da6ac-09fe-4b55-8
nnections': '5', 'device-model': 'vedge-cloud', 'version': '18.4.0
': '30', 'ompPeers': '2', 'latitude': '37.666684', 'longitude': '-
 'x86_64', 'uptime-date': 1555057800000, 'statusOrder': 4, 'device
', 'state_description': 'All daemons up', 'model_sku': 'None', 'lo
2', 'linux_cpu_count': '1', 'testbed_mode': False, 'layoutLevel':
======================
vManage-1   69DE12CE976BC070D5184152E0298516   18.4.0   11
vSmart-1   07F4E30F8E9368C48C06601C5AB51980   18.4.0   13
vSmart-2   1AA04C7DE9392B380677A0E7D1EFB055   18.4.0   17
vSmart-3   06D558D8F3630DCB1B81404B9D072529   18.4.0   11
vBond-1   4B1374A63640B43AFB55D49D4B2B35E8   18.4.0   --
vBond-2   24F41FC653922C582032704B260DEECC   18.4.0   --
DC-vE5   F0EFA950   18.4.0   5
DC-vE6   C29CC557   18.4.0   5
DC-vE7   9710283D   18.4.0   5
DC-vE8   E6E2EE3A   18.4.0   5
branch-vE1   D35CFDB3   18.4.0   5
branch-vE2   8F294336   18.4.0   5
branch-vE3   F0C2539C   18.4.0   5
branch-vE4   47652BC5   18.4.0   5
(hello)
```

Step 13. **MISSION**: Take few minutes and add some other pieces of data to the output like the "**state**", "**system-ip**" and "**personality**" so the output looks like the following.

```
======================
vManage-1   69DE12CE976BC070D5184152E0298516   18.4.0   11   green   172.1.0.18   vmanage
vSmart-1   07F4E30F8E9368C48C06601C5AB51980   18.4.0   13   green   172.1.0.16   vsmart
vSmart-2   1AA04C7DE9392B380677A0E7D1EFB055   18.4.0   17   green   172.1.0.17   vsmart
vSmart-3   06D558D8F3630DCB1B81404B9D072529   18.4.0   11   green   172.1.0.19   vsmart
vBond-1   4B1374A63640B43AFB55D49D4B2B35E8   18.4.0   --   green   172.1.0.15   vbond
vBond-2   24F41FC653922C582032704B260DEECC   18.4.0   --   green   172.1.0.20   vbond
DC-vE5   F0EFA950   18.4.0   5   green   172.1.100.5   vedge
DC-vE6   C29CC557   18.4.0   5   green   172.1.100.6   vedge
DC-vE7   9710283D   18.4.0   5   green   172.1.200.7   vedge
DC-vE8   E6E2EE3A   18.4.0   5   green   172.1.200.8   vedge
branch-vE1   D35CFDB3   18.4.0   5   green   172.1.10.1   vedge
branch-vE2   8F294336   18.4.0   5   green   172.1.20.2   vedge
branch-vE3   F0C2539C   18.4.0   5   green   172.1.30.3   vedge
branch-vE4   47652BC5   18.4.0   5   green   172.1.30.4   vedge
(hello)
admin@DESKTOP-IH4KQU5 MINGW64 /c/SDWAN-Programmability (master)
```

Step 14. Lets take the data and print it in columns with some labels next. Navigate to the desktop the SDWAN folder and open the file name **PythonLab6Source.txt** with Atom and copy entire contents to your clipboard.

```
for item in json_string['data']:
    system_ip=item['system-ip']
    print('{0:15}  {1:20}  {2}     {3:36}  '.format("System IP", "Hostname", "Version","UUID"))
    print('{0:15}  {1:20}  {2}     {3:36}  '.format("---------", "--------", "-------","------------------------------------"))
    print('{0:15}  {1:20}  {2}     {3:36}  '.format(item['system-ip'],item['host-name'],item['version'],item['uuid']))
```

Step 15. Paste the contents of the clipboard into the **PythonLab6.py script** that you are working on and place it below the previous lines you added. This section of code is yet another way to format the resulting json data and display it in a more friendly way.

```
for item in json_string['data']:
   print (item['host-name']+"   "+item['board-serial']+"   "+item['version']+"   "+item['controlConnections']+"

for item in json_string['data']:
    system_ip=item['system-ip']
    print('{0:15}  {1:20}  {2}     {3:36}  '.format("System IP", "Hostname", "Version","UUID"))
    print('{0:15}  {1:20}  {2}     {3:36}  '.format("---------", "--------", "-------","----------------------
    print ('{0:15}  {1:20}  {2}      {3:36}  '.format(item['system-ip'],item['host-name'],item['version'],item['
```

Step 16. Save the script as **PythonLab6** and then run the script in the terminal, the results should look like the following:

```
DC-vE8   E6E2EE3A   18.4.0   5   green   172.1.200.8   vedge
branch-vE1   D35CFDB3   18.4.0   5   green   172.1.10.1   vedge
branch-vE2   8F294336   18.4.0   5   green   172.1.20.2   vedge
branch-vE3   FOC2539C   18.4.0   5   green   172.1.30.3   vedge
branch-vE4   47652BC5   18.4.0   5   green   172.1.30.4   vedge
System IP        Hostname              Version     UUID
---------        --------              -------     ------------------------------------
172.1.0.18       vManage-1             18.4.0      618c3989-5dbe-4f92-95b8-cca81d1a466e
System IP        Hostname              Version     UUID
---------        --------              -------     ------------------------------------
172.1.0.16       vSmart-1              18.4.0      6e4d0357-e883-4931-853e-25e5a85ad979
System IP        Hostname              Version     UUID
---------        --------              -------     ------------------------------------
172.1.0.17       vSmart-2              18.4.0      34d16db3-ba55-43b1-a335-ae30e9b738ef
System IP        Hostname              Version     UUID
---------        --------              -------     ------------------------------------
172.1.0.19       vSmart-3              18.4.0      e7b3357b-a637-40f7-bf8b-d44742a44751
System IP        Hostname              Version     UUID
---------        --------              -------     ------------------------------------
172.1.0.15       vBond-1               18.4.0      8bc7e303-70d5-4bfe-b9ea-3dbf668f69cf
System IP        Hostname              Version     UUID
---------        --------              -------     ------------------------------------
172.1.0.20       vBond-2               18.4.0      c1515646-2b55-48ba-9341-1e908940e911
System IP        Hostname              Version     UUID
---------        --------              -------     ------------------------------------
```

Step 17. If you have errors you have 2 options, the first is to try and debug your script, the second is to go to the SDWAN folder on your desktop and use **File → Open** and open up the **PythonLab6Finished.py** script, save it as **PythonLab6.py** and run it.

Step 18. Do not proceed to any further labs until you have a functional script as this script will be used as the starting point for the next lab.

Step 19. End of lab

.

Lab 7 – Using the vManage REST API to Monitor the Deployment

To access the vManage GUI and REST API use the http addresses listed below.

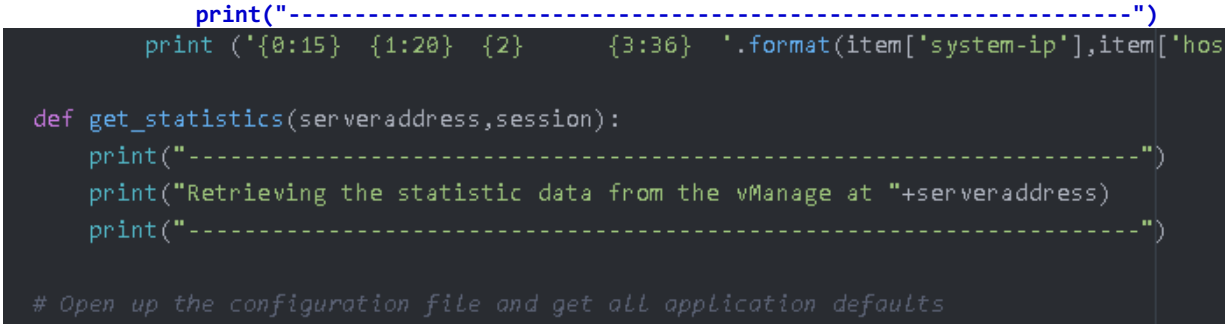
Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

Step 1. From the remote desktop of your admin PC navigate to the start button and go to **Start → Atom**

Step 2. In Atom use the **File → Open** menu and select the finished script your from lab 6 called **PythonLab6.py**. This will be the starting point for this lab.

Step 3. Before the section that starts with "**# Open up the configuration file and get all application defaults**", towards the bottom, enter the following lines right **above**:

```
def get_statistics(serveraddress,session):
    print("-----------------------------------------------------------------")
    print("Retrieving the statistic data from the vManage at "+serveraddress)
    print("-----------------------------------------------------------------")
```



Step 4. Now you will define the REST API URL that was used in lab 6 again which will be used to make a header for the statistics, copy and paste if you like. Put the lines of code below the previous step (ensure the indents match exaxtly the screenshot below when done):

```
    url = "https://" + serveraddress + "/dataservice/device"
    response = session.request("GET", url,verify=False, timeout=10)
    json_string = response.json()
```

```
def get_statistics(serveraddress,session):
    print("------------------------------------------------------------------------")
    print("Retrieving the statistic data from the vManage at "+serveraddress)

    url = "https://" + serveraddress + "/dataservice/device"
    response = session.request("GET",url,verify=False,timeout=10)
    json_string = response.json()

# Open up the configuration file and get all application defaults
```

Step 5. Copy the last section of the script built in lab 6 (it is 5 lines of code) to your clipboard.

```
    for item in json_string['data']:
        system_ip=item['system-ip']
        print('{0:15}  {1:20}  {2}     {3:36}  '.format("System IP", "Hostname", "Version","UUID"))
        print('{0:15}  {1:20}  {2}     {3:36}  '.format("---------", "--------", "-------","--------------------------------
        print ('{0:15}  {1:20}  {2}      {3:36}  '.format(item['system-ip'],item['host-name'],item['version'],item['uuid']))
```

Step 6. Paste the text that is on your clipboard below the URL lines you just added. You will use this as the header for your statistics.

```
    url = "https://" + serveraddress + "/dataservice/device"
    response = session.request("GET",url,verify=False,timeout=10)
    json_string = response.json()

    for item in json_string['data']:
        system_ip=item['system-ip']
        print('{0:15}  {1:20}  {2}     {3:36}  '.format("System IP", "Hostname", "Version","UUID"))
        print('{0:15}  {1:20}  {2}     {3:36}  '.format("---------", "--------", "-------","------------------------------------"))
        print ('{0:15}  {1:20}  {2}      {3:36}  '.format(item['system-ip'],item['host-name'],item['version'],item['uuid']))
```

Step 7. Next you will add the statistics to the scripts by invoking another REST API. Enter the following lines to the script, indented under the previous section

```
print ("Retrieving Statistics for "+system_ip)
url = "https://"+serveraddress+"/dataservice/device/interface/stats?deviceId="+system_ip
response = session.request("GET", url,verify=False)
json_string=response.json()
```

```
    for item in json_string['data']:
        system_ip=item['system-ip']
        print('{0:15}  {1:20}  {2}     {3:36}  '.format("System IP", "Hostname", "Version","UUID"))
        print('{0:15}  {1:20}  {2}     {3:36}  '.format("---------", "--------", "-------","--------
        print ('{0:15}  {1:20}  {2}      {3:36}  '.format(item['system-ip'],item['host-name'],item['

        print ("Retrieving Statistics for "+system_ip)
        url = "https://"+serveraddress+"/dataservice/device/interface/stats?deviceId="+system_ip
        response = session.request("GET", url,verify=False)
        json_string=response.json()
```

Step 8. Add the following 2 lines to the script below the last line you added:

```
rx=0
tx=0
print(json_string)
```

```
        print ("Retrieving Statistics for "+system_ip)
        url = "https://"+serveraddress+"/dataservice/device/interface/stats?deviceId="+system_ip
        response = session.request("GET", url,verify=False)
        json_string=response.json()
        rx=0
        tx=0
        print(json_string)
```

Step 9. At the very bottom of the script add an indented line **get_statistics(serveraddress,session)**

```
session=initalize_connection(serveraddress,username,password)
if session != False:
    print ("Successful you will issue API commands here in later labs")
    get_inventory(serveraddress,session)
    get_statistics(serveraddress,session)
```

Step 10. Save the script as PythonLab7.py and then run the script using the terminal, the results should have sections that look like the following, notice the headers separate the statistics for the different devices in the SDWAN deployment.

```
System IP        Hostname            Version     UUID
---------        --------            -------     ------------------------------------
172.1.30.4       branch-vE4          18.4.0      276da6ac-09fe-4b55-89f3-ec8fd833fd0c
Retrieving Statistics for 172.1.30.4
{'header': {'generatedOn': 1555418595466, 'viewKeys': {'uniqueKey': ['vdevice-dataKey'], 'preferenceKey': 'grid-Inte
rfaceStats'}, 'columns': [{'title': 'VPN', 'property': 'vpn-id', 'width': 50, 'dataType': 'string'}, {'title': 'Inte
rface', 'property': 'ifname', 'width': 100, 'dataType': 'string'}, {'title': 'AF Type', 'property': 'af-type', 'widt
h': 100, 'dataType': 'string'}, {'title': 'Rx Pkts', 'property': 'rx-packets', 'width': 100, 'dataType': 'string'},
{'title': 'Rx Octets', 'property': 'rx-octets', 'width': 100, 'dataType': 'string'}, {'title': 'Rx Errors', 'propert
y': 'rx-errors', 'width': 100, 'dataType': 'string'}, {'title': 'Rx Drops', 'property': 'rx-Drops', 'width': 100, 'd
ataType': 'string'}, {'title': 'Tx Pkts', 'property': 'tx-packets', 'width': 100, 'dataType': 'string'}, {'title': '
Tx Octets', 'property': 'tx-octets', 'width': 100, 'dataType': 'string'}, {'title': 'Tx Errors', 'property': 'tx-err
ors', 'width': 100, 'dataType': 'string'}, {'title': 'Tx Drops', 'property': 'tx-drops', 'width': 100, 'dataType': '
string'}, {'title': 'Rx PPS', 'property': 'rx-pps', 'width': 100, 'dataType': 'string'}, {'title': 'Rx Kbps', 'prope
rty': 'rx-kbps', 'width': 100, 'dataType': 'string'}, {'title': 'Tx PPS', 'property': 'tx-pps', 'width': 100, 'dataT
```

Step 11. Now comment out the **print (json_string**) by placing a # in front of it and add the following lines below.

rx=0
tx=0
#print(json_string)

```
        print ("Retrieving Statistics for "+system_ip)
        url = "https://"+serveraddress+"/dataservice/device/interface/stats?deviceId="+system_ip
        response = session.request("GET", url,verify=False)
        json_string=response.json()
        rx=0
        tx=0
        #print(json_string)
```

Step 12. Lets make some of the data a bit easier to read by adding some text from the **PythonLab7Source1.txt** file in the SDWAN folder on your desktop. Copy the contents of this file and paste it below the commented out **#print(json_string)**. Make sure the indentations match the screenshot below.

```
        try:
            for stats in json_string['data']:
                # We only want to print the ipv4 interfaces
                if stats['af-type']!='ipv6':
                    print('      {0:9}     {1:10}   {2:10d}      {3:10d}'.format(stats['ifname'],stats['vpn-id'],int(stats['tx-packets']),i
                rx=rx+int(stats['rx-packets'])
                tx=tx+int(stats['tx-packets'])
            print('                                {0}      {1}'.format("----------","----------"))
            print('      {0:9}     {1:10}   {2:10d}      {3:10d}'.format(" ","Total",tx,rx))
        except KeyError:
            print('Data not found for that System IP')
    print ("\n")

```

```
        #print(json_string)
        try:
            for stats in json_string['data']:
                # We only want to print the ipv4 interfaces
                if stats['af-type']!='ipv6':
                    print('      {0:9}     {1:10}   {2:10d}      {3:10d}'.format(stats['ifname'],stats['vpn-id'],int
                rx=rx+int(stats['rx-packets'])
                tx=tx+int(stats['tx-packets'])
            print('                                {0}      {1}'.format("----------","----------"))
            print('      {0:9}     {1:10}   {2:10d}      {3:10d}'.format(" ","Total",tx,rx))
        except KeyError:
            print('Data not found for that System IP')
    print ("\n")
```

Step 13. Save the script as PythonLab7 and then run the script using the **terminal**, the resulting sections for the devices in the SDWAN deployment should look like the following:

```
System IP        Hostname             Version     UUID
---------        --------             -------     ------------------------------------
172.1.30.3       branch-vE3           18.4.0      e9e59d4a-23b9-4eb3-a2a6-0fe7647412a9
Retrieving Statistics for 172.1.30.3
      ge0/0          0              11972462        15502075
      ge0/1          0                     0               2
      ge0/3          0                     9         3549486
      system         0                     0               0
      ge0/1.30       0               5996820         5991827
      ge0/1.40       0               5022713         5001593
      ge0/2          1                     7         3549448
      eth0           512                  25        31686762
                                    ----------      ----------
                     Total          22992062        65295382
System IP        Hostname             Version     UUID
---------        --------             -------     ------------------------------------
172.1.30.4       branch-vE4           18.4.0      276da6ac-09fe-4b55-89f3-ec8fd833fd0c
Retrieving Statistics for 172.1.30.4
      ge0/1          0                     0               1
      ge0/2          0              10042316        13511081
      ge0/3          0                     9         3549424
      system         0                     0               0
      ge0/1.30       0               5991874         5996823
      ge0/1.40       0               5001615         5022715
      ge0/0          1                 38566         3510893
      eth0           512                  27        31686702
                                    ----------      ----------
                     Total          21074434        63291784
```

Step 14. From the SDWAN folder on your desktop open the **PythonLab7Source2.txt** file and use CTRL+A to hightlight the entire contents and then copy it to your clipboard.

```
def get_tunnel_statistic(serveraddress,session,systemip):

    """
    This function will return the details for all the tunnels for a particular endpoint
    :param serveraddress: This is the IP Address and Port number of vManage (i.e., "192.168.0.1:8443")
    :param session: session object that was returned from the prior initialize_connection function
    :param systemip: systemip of a device that we want to receive detailed statistics on
    :return: nothing
    """
    print ("Returning the tunnel statistics for device: "+systemip+"\n")

    url = "https://"+serveraddress+"/dataservice/device/tunnel/statistics?deviceId="+systemip

    response = session.request("GET", url,verify=False,timeout=10)
    json_string=response.json()
    #print (json_string)
    #for item in json_string['data']:
    #    print(item)

    # If there is an error, with the query then let's print out the error code
    if 'error' in json_string:
        print("An Error Occured processing the data")
        print(json_string['error']['details'])
    else:
```

Step 15. Paste the contents of you clipboard right above the section that starts with "**# Open up the configuration file and get all application defaults**" (below where you were just working.

```
    print ("\n")

def get_tunnel_statistic(serveraddress,session,systemip):

    """
    This function will return the details for all the tunnels for a particular endpoint
    :param serveraddress: This is the IP Address and Port number of vManage (i.e., "192.168.0.1:8443")
    :param session: session object that was returned from the prior initialize_connection function
    :param systemip: systemip of a device that we want to receive detailed statistics on
    :return: nothing
    """
    print ("Returning the tunnel statistics for device: "+systemip+"\n")

    url = "https://"+serveraddress+"/dataservice/device/tunnel/statistics?deviceId="+systemip
```

Step 16. At the very bottom of the script enter the following line:
get_tunnel_statistic(serveraddress,session,systemip)

```
session=initalize_connection(serveraddress,username,password)
if session != False:
    print ("Successful you will issue API commands here in later labs")
    get_inventory(serveraddress,session)
    get_statistic(serveraddress,session)
    get_tunnel_statistic(serveraddress,session,systemip)
```

Step 17. Save the script as **PythonLab7.py** and then run the script using the terminal run the python script, the results should look like the following:

```
Returning the tunnel statistics for device: 172.1.10.1

     Tunnel                  Color          RX Pkts    TX Pkts     RX Bytes     TX Bytes
     ------                  -----          -------    -------     --------     --------
branch-vE1->branch-vE2     gold             2065983    2065983   2773016560   2773009152
branch-vE1->branch-vE3     gold             2066007    2066007   2773048720   2773041312
branch-vE1->branch-vE4     gold             2066041    2066047   2773496352   2773496840
branch-vE1->DC-vE5         gold             2066327    2066327   2775385400   2775379552
branch-vE1->DC-vE6         gold             2066021    2066022   2773117512   2773108400
branch-vE1->DC-vE7         gold             2065694    2065829   2772685744   2772864296
branch-vE1->DC-vE8         gold              524512     524512    704549912    704538888
branch-vE1->branch-vE2     silver           2065673    2065802   2772543720   2772730672
branch-vE1->branch-vE3     silver           2065672    2065809   2772542352   2772740304
branch-vE1->branch-vE4     silver           2065666    2065794   2772531064   2772695320
branch-vE1->DC-vE5         silver           2065960    2066078   2774407504   2774613384
branch-vE1->DC-vE6         silver           2065672    2065809   2772542352   2772740304
branch-vE1->DC-vE7         silver           2065679    2065786   2772552176   2772699888
branch-vE1->DC-vE8         silver            524512     524512    704549912    704538888
                                                                  ----------   ----------
                                            Totals:              34686969280  34688197200
```

Step 18. End of lab

Lab 8 – Using the vManage REST API to Manage and Configure SDWAN devices

To access the vManage GUI and REST API use the http addresses listed below.

Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

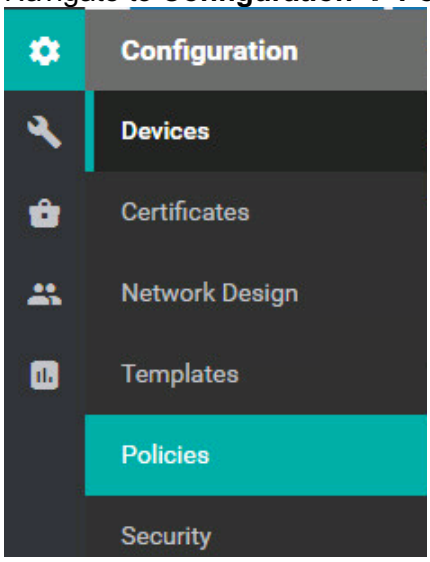
Case Study 1

Company XYZ wants to route FTP traffic differently based on time of day. However there is no way in the GUI to vManage to accomplish this. You will use the REST API and a Python script to activate/deactivate a vSmart policy that is applied based on a schedule. In this lab you will use the REST API to activate and deactivate a predefined policy that will route traffic over a specific path as opposed to load balanced when the policy is not applied.

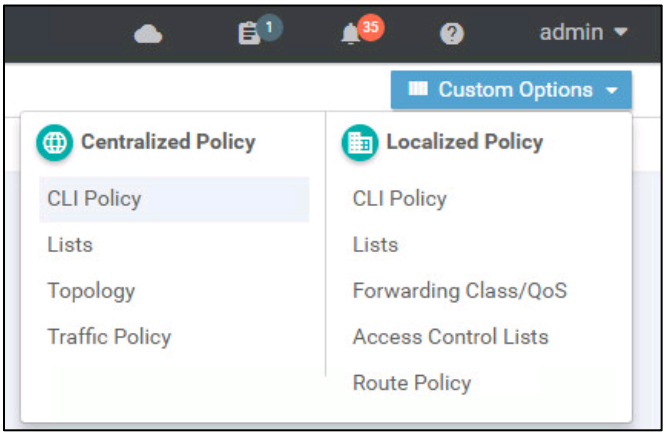
Step 1. First you will need to create a policy that will route FTP traffic differently when it is applied. Open Chrome and navigate to the vManage GUI.

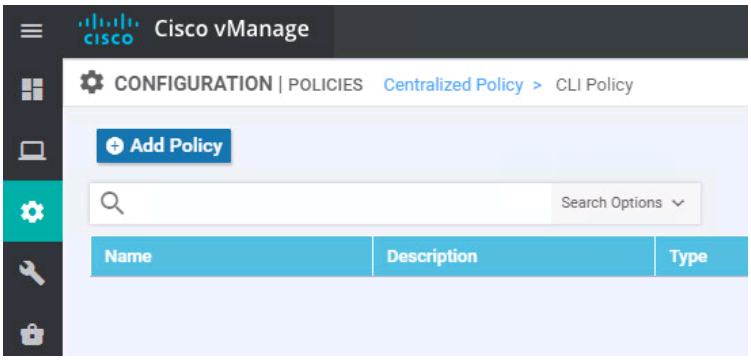
Step 2. Login when prompted.

Step 3. Navigate to **Configuration → Policies**



Step 4. In the upper right hand corner select use the **Custom Options** button and select **Centralized** a **CLI Policy**

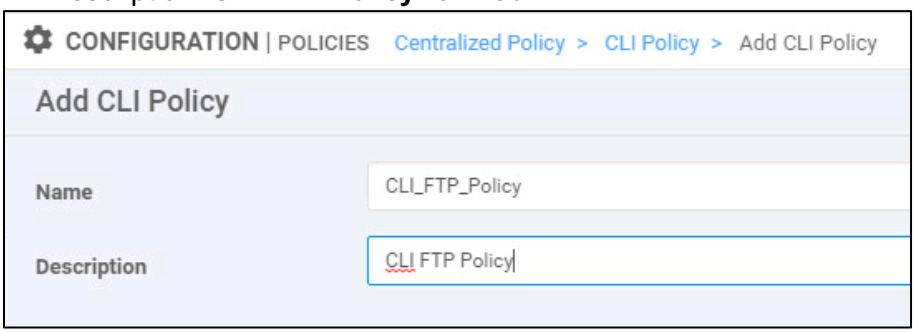




Step 5. Press the +**Add Policy** button to add a new CLI Policy.

Step 6. Fill out the CLI Policy page with the following:

- Name: **CLI_FTP_Policy_PodXX**
- Description: **CLI FTP Policy for PodXX**

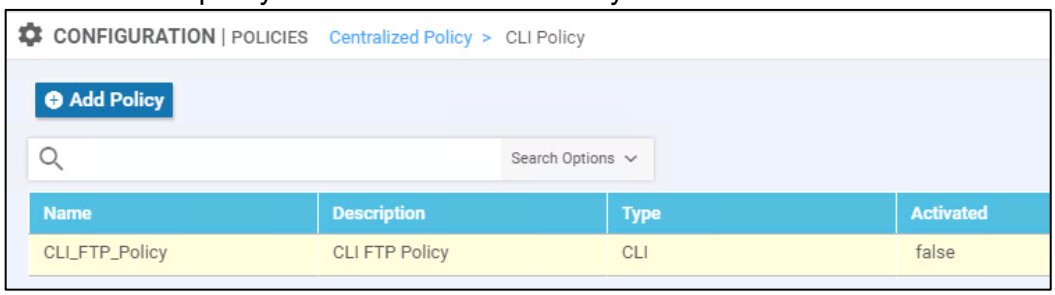


Step 7. Open the file called **FTPPolicy.txt** in the folder called SDWAN Lab Files, copy the entire contents to your clipboard and paste into the CLI Configuration field on the CLI Policy page. Then press the **Add** button.

CLI Configuration

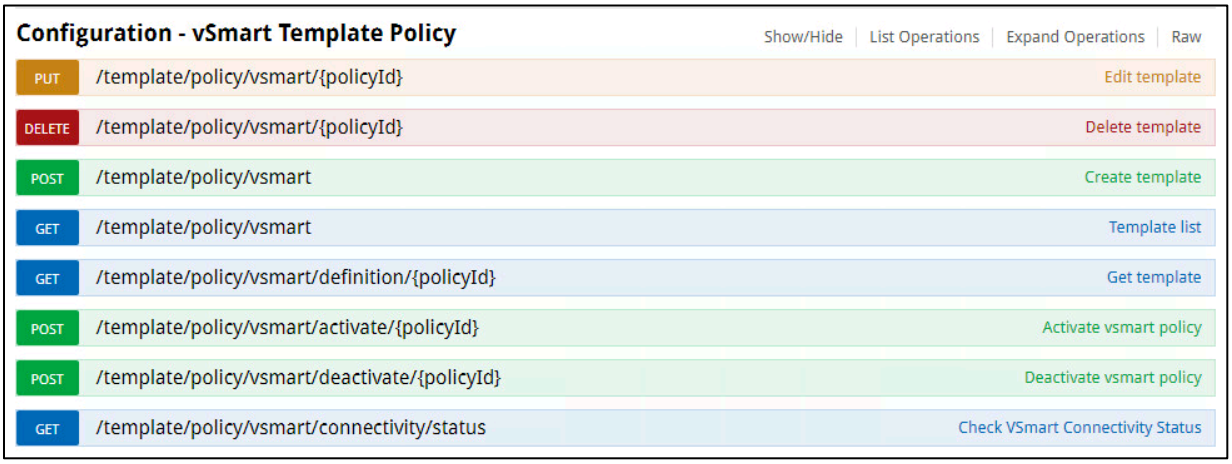
```
policy
 data-policy _VPN1_FTP_Off_Peak
  vpn-list VPN1
    sequence 1
     match
      app-list FTP_traffic
      source-ip 0.0.0.0/0
     !
     action accept
      set
       local-tloc-list
        color gold
        encap ipsec
      !
     !
    !
  default-action accept
 !
 lists
  app-list FTP_traffic
   app ftp
   app ftp_data
   app ftp-data
  !
  site-list SiteBranch
   site-id 10
   site-id 20
   site-id 30
  !
  site-list SiteDC
   site-id 100
   site-id 200
  !
  vpn-list VPN1
   vpn 1
  !
 !
!
apply-policy
 site-list SiteBranch
  data-policy _VPN1_FTP_Off_Peak all
 !
 site-list SiteDC
  data-policy _VPN1_FTP_Off_Peak all
 !
!
```

Step 8. Notice that the policy Activated state is currently **false**.

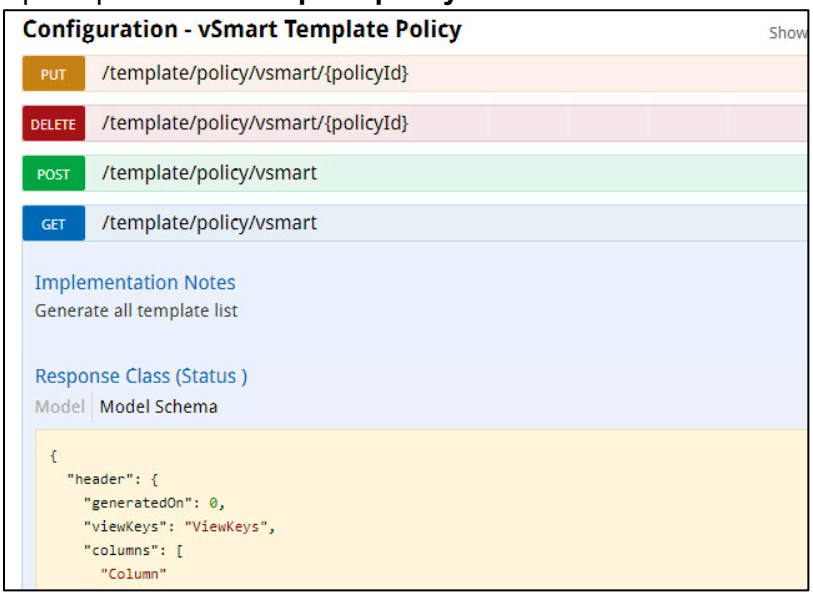


Step 9. Now you need to find out what the policy ID is, this can be done thru the API. Open up the API documentation by going to **https://1.4.28.28:8443/apidocs**

Step 10. Click on the **Configuration – vSmart Template Policy** to expand the API calls.



Step 11. Open up the **GET /template/policy/vsmart** section



Step 12. Press the **Try it Out!** button and view the response body and look for the **policyID** towards the bottom of the output for a policy that your instructor has made called Lab8_Policy. The policyID **will be different** than the one listed in the screenshot below.

Step 13. Note the policyID as this will be needed later in the lab, leave this page open so you can copy the policyID later.

Next you will open a python script that will be used to activate and deactivate the policy on the vSmarts VM's. This script will activate the Policy for 10 seconds then will deactivate the policy for 10 seconds, this cycle will repeat 2 times. This script could be modified to activate the policy at a particular time of day, day of the week, or month.

Step 14. Open the script called **PythonLab8Source.py** using Atom.

```python
#!/usr/bin/env python
"""This Python Script will activate a script and then 5 seconds deactivate a script
"""



import requests
import urllib3
import datetime
import time
# Disable Certificate warning
try:
    requests.packages.urllib3.disable_warnings()
except:
    pass
from requests.auth import HTTPBasicAuth
import sys

# Get the absolute path for the directory where this file is located "here"
here = os.path.abspath(os.path.dirname(__file__))

# Get the absolute path for the project / repository root
project_root = os.path.abspath(os.path.join(here, '../..'))

# Extend the system path to include the project root and import the env files
sys.path.insert(0, project_root)
```

Step 15. Go back the Test API call you did and copy the policyID to your clipboard being careful to **NOT** copy the quotes.

Step 16. On Line 68, 69, and 70 replace **PUT_YOUR_POLICY_ID_HERE** with the policyID on your clipboard.

```python
def policy_cycle(sdwan_host,session):
    policy_id = "PUT_YOUR_POLICY_ID_HERE"
    activate_policy_url = "https://%s/dataservice/template/policy/vsmart/activate/PUT_YOUR_POLICY_ID_HERE"%(sdwan_host)
    deactivate_policy_url = "https://%s/dataservice/template/policy/vsmart/deactivate/PUT_YOUR_POLICY_ID_HERE"%(sdwan_host)
    payload = "{\n }"
    headers = {'Content-Type':'application/json'}
```

```python
def policy_cycle(sdwan_host,session):
    policy_id = "c19aa591-3993-4f67-a76c-5194b368345fE"
    activate_policy_url = "https://%s/dataservice/template/policy/vsmart/activate/c19aa591-3993-4f67-a76c-5194b368345f"%(sdwan_host)
    deactivate_policy_url = "https://%s/dataservice/template/policy/vsmart/deactivate/c19aa591-3993-4f67-a76c-5194b368345f"%(sdwan_host)
```

Step 17. Save the script as **PythonLab8.py** and then use terminal to run the python script using the command **python3.7 PythonLab8.py**.

Step 18. Wait for 20 seconds for the script to finish and review the output. Notice that the policy is activated and deactivated twice.

```
(sdwan) ubuntu@zzDNA:~/SDWAN-Programmability$ python3.7 PythonLab8.py
SDWAN Login to 1.4.28.28 as admin ...
Connection to vManage successful
Policy Activated for 5 seconds @ 2019-07-11 20:02:52.078848
Policy Deactivated for 5 seconds @ 2019-07-11 20:02:57.091561
Policy Activated for 5 seconds @ 2019-07-11 20:03:02.102287
Policy Deactivated for 5 seconds @ 2019-07-11 20:03:07.110967
```

Step 19. Take some time and look at the python scripting, notice the scripting code used is a bit different than our previous scripts. Multiple ways to the same end result.

Step 20. Where is the script getting credentials to login to the API from.

Step 21. How does the script determine how many times to activate or deactivate the policy

Step 22. **Bonus Mission (Easy):** Change the message displayed when the policy is activated and include a carriage return

Step 23. **Bonus Mission (Advanced):** Make two scripts one to activate the policy and one to deactivate the policy. Then create a cron job for each to run the activate script followed by the deactivate 2-3 minutes later. (Google is your friend if you are unfamiliar with cron)

Step 24. Deactive your virtual environment when done with this lab, as you will recreate it in the next lab (use the **deactivate** command).

Step 25. End of lab

Lab 9 – Posting to Webex Teams when a Policy is Activated or Deactivated

To access the vManage GUI and REST API use the http addresses listed below.

Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

Case Study 2

Company XYZ wants to know when a policy change has occurred and they want to be notified in the Webex Teams support space (our classroom space) . In this lab you will configure a python script to post a message to the Webex Teams space whenever the time of day policy is activated or deactivated

Step 1. First you will need to get some Webex Teams information. You will need the following:
- Webex Teams Classroom Space ID (provided by your instructor)
- Your Webex Teams Token

Step 2. Ask your instructor to post the Webex Teams Classroom Space into the Webex Teams space so you can easly copy it.

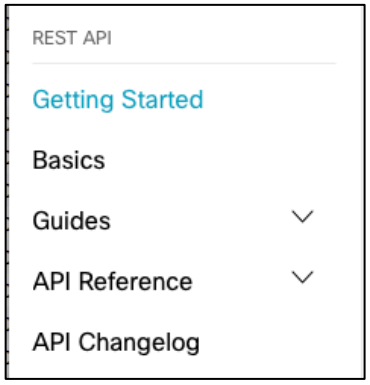
NOTE: If you need help copying and pasting from your PC to the Remote desktop please ask your instructor to assist you.

Step 3. Copy the ID to your clipboard, this value will need to be copied into a text file found in your cloned repository **SDWAN-Programmability\env_user.py**. Open this file and in the copy it into the value for SPARK_ROOM_ID

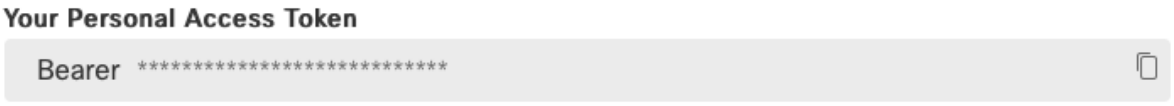
```
35  # User Input
36
37  SPARK_ACCESS_TOKEN = "Get from developer.webex.com --> Documentation --> Getting Started"
38  SPARK_ROOM_ID = "Your Instructor will provide"
39
```

Step 4. Next open a web browser on your Admin PC and go to https://developer.webex.com and login with your Webex Teams credentials.

Step 5. Select **Documentation** at the top of the page, then on the left hand side select **Getting Started**.



Step 6. In the middle of the resulting page you will see **Your Personal Access Token**, this is good for 12 hours only and will then be changed by Cisco. So if you come back to labs later you will have to repeat the following 3 steps.



Step 7. Copy the Token by clicking on the icon on the right of the Token field

Step 8. Paste this token into the **env_user.py** field call **Spark_Access_Token**

```
35  # User Input
36
37  SPARK_ACCESS_TOKEN = "Yjg1NWMwMjAtMTE2OC00N2RhLTg0MjEtNGFjYTA4M2MzZmM4NmZlODEzNzYtZTI4_PF84_1eb65fdf-9643-417f-9974-ad72cae0e10f"
38  SPARK_ROOM_ID = "Y2lzY29zcGFyazovL3VzL1JPT00vZDU1MTQ5ZTAtNjBjYy0xMWU5LThjYzEtMmRlNWIwZDY2MGNi"
39
```

Step 9. Save the **env_user.py** file.

Step 10. Open up in Atom the finished file from Lab 8 called **PythonLab8.py**

Step 11. Save a copy of this script as PythonLab9.py

Step 12. Add the following code to the import the ciscosparkapi module in the python script.

```
import ciscosparkapi
```

```
5   import ciscosparkapi
6   import json
7   import os
8   import requests
9   import urllib3
10  import datetime
11  import time
```

Step 13. At the bottom of the import statements enter the following command:

```
import env_user
```

```
27  sys.path.insert(0, project_root)
28  import env_lab
29  import env_user
```

Step 14. Below the import statemments enter the following lines to set a variable named spark that will be set to your access token:

```
# Create a Cisco Spark (Webex Teams) object
spark = ciscosparkapi.CiscoSparkAPI(access_token=env_user.SPARK_ACCESS_TOKEN)
```

```
31  # Create a Cisco Spark (Webex Teams) object
32  spark = ciscosparkapi.CiscoSparkAPI(access_token=env_user.SPARK_ACCESS_TOKEN)
```

Step 15. Under the **policy_cycle** definition find the "**print ("Policy Activated 5 sec@ %s "%t)** and enter the following line under it (mind the indentations):

```
message = spark.messages.create(env_user.SPARK_ROOM_ID,text='Look No Hands!')
print(message)
print('\n\n')
```

```
    while i<2:
        i+=1
        response = session.request("GET",activate_policy_url,verify=False,timeout=10)
        data = response.content
        #print (data)
        t = datetime.datetime.now()
        print ("Policy Activated for 5 seconds @ %s "%t)
        message = spark.messages.create(env_user.SPARK_ROOM_ID,text='Look No Hands!')
        print(message)
        print('\n\n')
        time.sleep(5)
```

Step 16. Repeat adding the 3 line in the deactivate section as well

Step 17. Save your **PythonLab9.py** script

Step 18. Open up a new terminal window and enter the following commands to setup a new virtual environment.

- `cd SDWAN-Programmability`
- `python3.7 -m venv sdwan`
- `source sdwan/bin/activate`

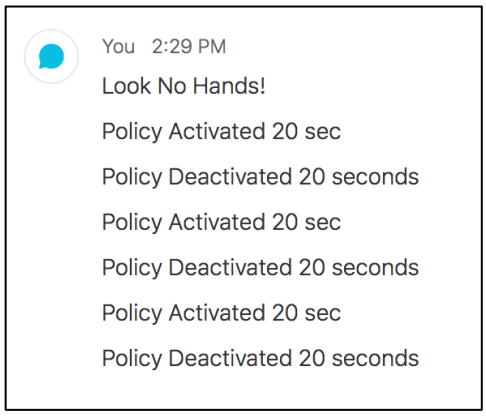
Step 19. Open from the SDWAN-Programmability directory the file called **requirements.txt**.

```
1  ciscosparkapi==0.10.1
2  requests==2.21.0
3  urllib3>=1.24.2
4  xmltodict==0.11.0
5  configparser==3.7.4
```

Step 20. Enter the command **pip install –r requirements.txt** to install all the required modules with one command.

Step 21. Open Webex Teams and go the Classroom space.

Step 22. Now run the script with the command **python3.7 PythonLab9.py** and view the output, wait for about 80 seconds for the script to finish. Verify that your see your output in the space.



Step 23. **Optional Mission (Advanced):** Using your mad python skills get the inventory of the SDWAN deployment and post it to the Webex Teams space in addition to the policy changes. Post your script to the chat room so everyone who could not figure this out has an example!

Step 24. End of Lab

Lab 10 – Opening a Ticket in ServiceNow

To access the vManage GUI and REST API use the http addresses listed below.

Device Name	vManage GUI and REST API	Username	Password
vManage-1	1.4.28.28	admin	admin

Case Study 3

Company XYZ wants to know when a policy change has occurred and they want to be notified in the Webex Teams support space (our classroom space) . In this lab you will configure a python script to post a message to the Webex Teams space whenever the time of day policy is activated or deactivated

Step 1. Steps 1 thru 16 may be skipped and you may use the instructors instance of ServiceNow. If you wish to make your own instance of Service now be aware that there may not be availability as the number of instances is limited by ServiceNow. Use the instructors in this case.

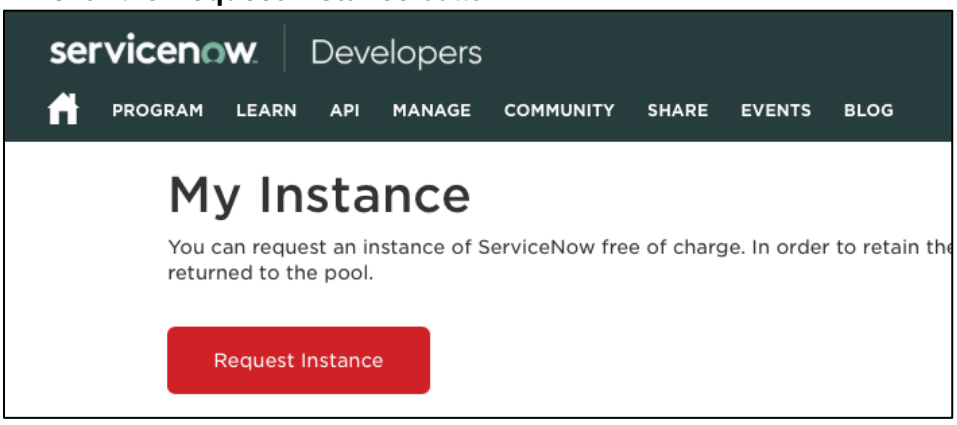
Step 2. Go to https://developer.servicenow.com and create an account (click Register)!

Step 3. After activating your account, login into the developer site.

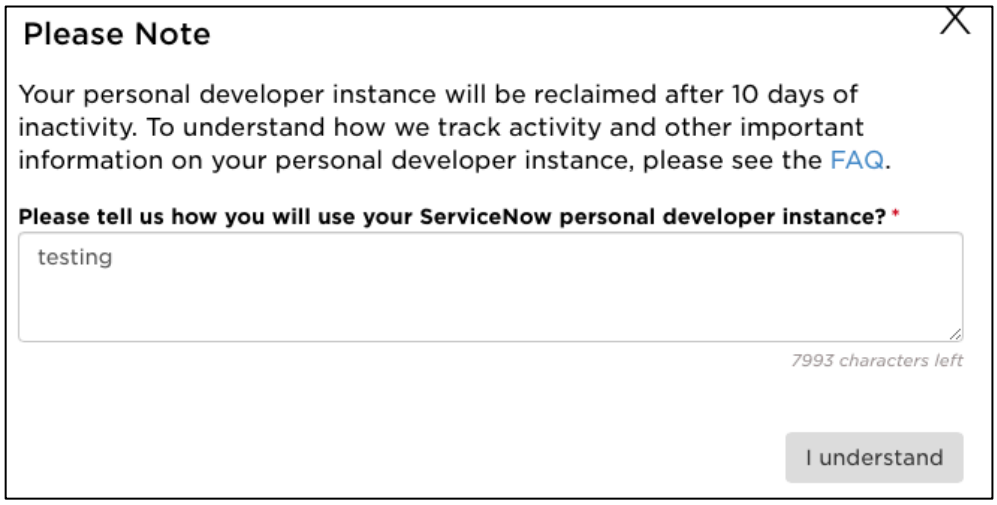
Step 4. Create Personal Developer Instance

Step 5. You will need to request an Instance on the ServiceNow web site. Obtaining a personal developer instance is fast and easy. Follow the steps below

- Login to developer.servicenow.com
- Open the **Manage** menu and click the **Instance** menu option
- Click the **Request Instance** button



Step 6. Enter "**testing**" and then the **I understand** button.



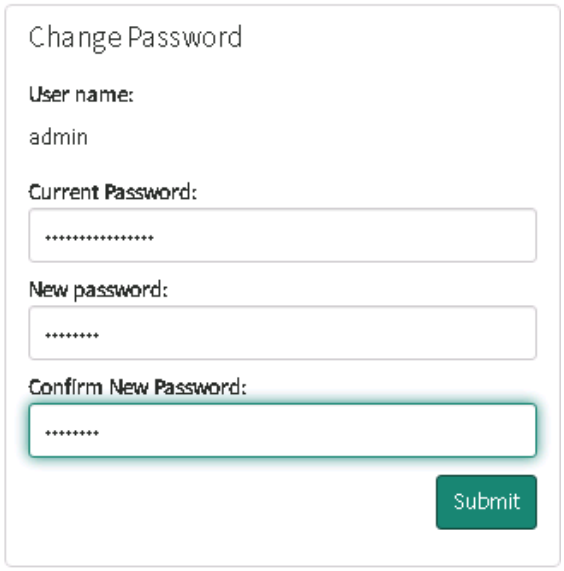
Step 7. Choose the instance version of **New York** (Madrid will not work with the script we will write).

Step 8. When the instance is assigned, the screen updates to display the instance URL and the admin credentials. If you navigate away from the Manage Instance page, you will receive your instance name and admin password by email. Copy the admin password to the clipboard. This will be needed if you want to log back into your developer instance later.

Step 9. Open Atom and copy the URL, username and password that you copied to your clipboard so we have it later if/when we need them.

Step 10. Click the link for you instance and use the username of admin and the password that you copied to Atom

Step 11. Change your admin password to C!sc0123



Step 12. Your Home page in ServiceNow should load

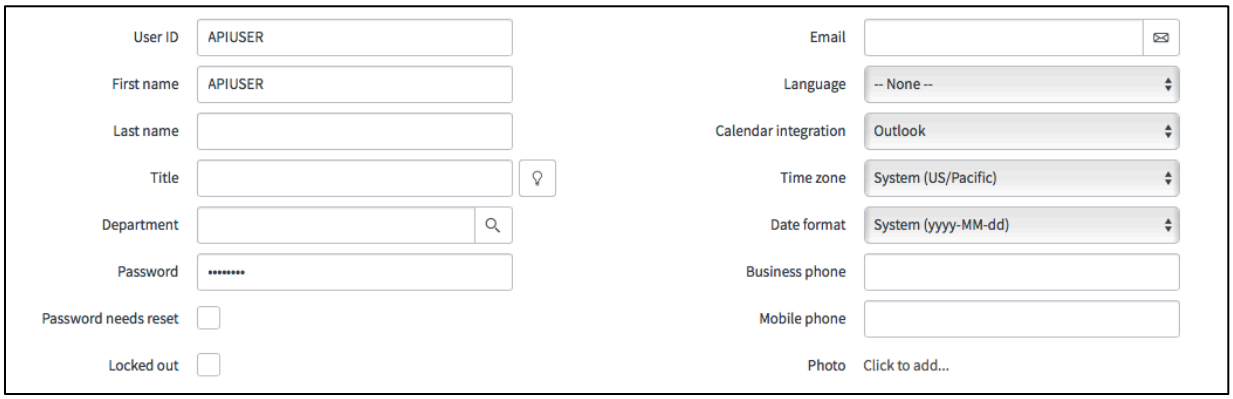
In order to keep the developer program free and provide PDIs to everyone who wants one, instances hibernate when they are idle. You can wake up a hibernating instance by navigating

to **Manage > Instance** on the ServiceNow developer site. It takes approximately three minutes to wake up a hibernating instance. Your work is saved on hibernating instances.
If there is no instance activity within 10 days, the instance is reclaimed. Reclaimed instances and their data cannot be recovered. Log in to your instance and use it regularly to keep the instance active.
You can read more about hibernation and instance activity on the ServiceNow developer site FAQ.

NOTE: If you need help copying and pasting from your PC to the Remote desktop please ask your instructor to assist you.

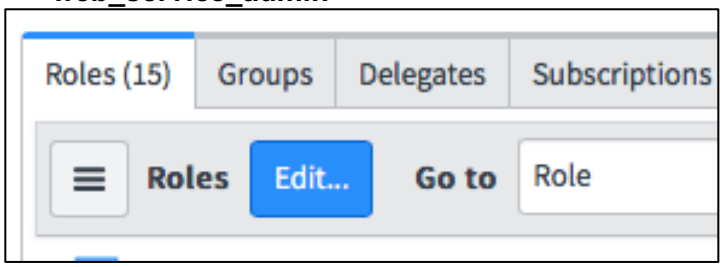
Step 13. Your Home page in ServiceNow should load, there is a grid of buttons that appears, select the **User Management** button.

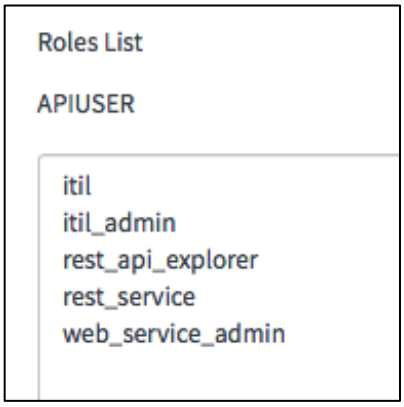
Step 14. Add a new user named **APIUSER** with a password of **C!sc0123** (that is a zero).



Step 15. On the Roles tab select the Edit button and assign the user the following roles which will allow this credential to be used to make REST API calls:

- **itil**
- **itil_admin**
- **rest_api_explorer**
- **rest_service**
- **web_service_admin**





Step 16. On the left hand side navigate to **Incident → Open**

Step 17. In the SDWAN-Programmablity folder in your home directory, open up the **env_lab.py** script in Atom.

```
# Where to connect and with what credentials
vManage = {
    "host": "1.4.28.28",
    "username": "admin",
    "password": "admin"
}
```

Step 18. Add the following section below the vManage section as follows, use the url that was pasted into notepad:

```
serviceNow = {
    "url": "YOUR_URL_FROM_INSTANCE/api/now",
    "username: "APIUSER",
    "password: "C!sc0123"
}
```

```
# Where to connect and with what credentials
vManage = {
    "host": "1.4.28.28",
    "username": "admin",
    "password": "admin"
}
serviceNow = {
    "url": "https://dev81865.service-now.com/api/now/",
    "username": "APIUSER",
    "password": "C!sc0123"
}
```

NOTE: Your url (devXXXX) will be different from the one shown in the screenshot

Step 19. Save the **env_lab.py** file.

Step 20. Now Open up the final version of your PythonLab9.py script in Atom and you will add to this script to log an incident after the policy on the vManage has bee changed and a message written to Webex Teams.

Step 21. Do a Save As of the script and name it **PythonLab10.py**, this will be the script you work on for this lab.

Step 22. Add the following section under the sdwan credentials:

```
# Details for ServiceNow API Access
snow_url = env_lab.serviceNow['url']
snow_user = env_lab.serviceNow['username']
snow_pass = env_lab.serviceNow['password']
```

Under

```
# Details for SDWAN Center Platform API calls from env_lab file
sdwan_host = env_lab.vManage['host']
sdwan_user = env_lab.vManage['username']
sdwan_pass = env_lab.vManage['password']
sdwan_headers = {'content-type': 'application/json'}

# Details for ServiceNow API Access
snow_host = env_lab.serviceNow['url']
snow_user = env_lab.serviceNow['username']
snow_pass = env_lab.serviceNow['password']
```

Step 23. In Atom open up **PythonLab10Source** in another tab.

```
1  def get_user_sys_id(snow_user):
2
3      # find the ServiceNow user_id for the specified user
4
5      url = snow_url + '/table/sys_user?sysparm_limit=1&name=' + snow_user
6      headers = {'Content-Type': 'application/json', 'Accept': 'application/json'}
7      response = requests.get(url, auth=(snow_user, snow_pass), headers=headers)
8      user_json = response.json()
9      return user_json['result'][0]['sys_id']
10
11
12 def create_incident(description, comment, snow_user, severity):
13
14     # This function will create a new incident with the {description}, {comments}, severity for the {user}
15
16     caller_sys_id = get_user_sys_id(snow_user)
17     print caller_sys_id
18     url = snow_urlL + '/table/incident'
19     payload = {'short_description': description,
20                'comments': (comment + ', \nIncident created using APIs by caller by PUT_YOUR_NAME_HERE'),
21                'caller_id': caller_sys_id,
22                'urgency': severity
23                }
24     headers = {'Content-Type': 'application/json', 'Accept': 'application/json'}
25     response = requests.post(url, auth=(snow_user, snow_pass), data=json.dumps(payload), headers=headers)
26     incident_json = response.json()
27     return incident_json['result']['number']
```

Replace

Step 24. In line 20 replace the **PUT_YOUR_NAME_HERE** with your actual name, then copy the entire contents to your clipboard

Step 25. Paste the entire contents of the clipboard before the **session=initialize connection(sdwan_host,sdwan_user,sdwan_pass)**

```
def get_user_sys_id(username):

    # find the ServiceNow user_id for the specified user

    url = snow_url + '/table/sys_user?sysparm_limit=1&name=' + snow_user
    #print(url)
    headers = {'Content-Type': 'application/json', 'Accept': 'application/json'}
    response = requests.get(url, auth=(snow_user,snow_pass), headers=headers)
    user_json = response.json()
    #print (user_json['result'][0]['sys_id'])
    return user_json['result'][0]['sys_id']


def create_incident(description, comment, snow_user, severity):

    # This function will create a new incident with the {description}, {comments}, severity for the {user}

    caller_sys_id = get_user_sys_id(snow_user)
    url = snow_url + '/table/incident'
    payload = {'short_description': description,
               'comments': (comment + ', \nIncident created using APIs by caller PUT_YOUR_NAME_HERE'),
               'caller_id': caller_sys_id,
               'urgency': severity
               }
    headers = {'Content-Type': 'application/json', 'Accept': 'application/json'}
    response = requests.post(url, auth=(snow_user, snow_pass), data=json.dumps(payload), headers=headers)
    incident_json = response.json()
    return incident_json['result']['number']

session=initalize_connection(sdwan_host,sdwan_user,sdwan_pass)
if session != False:
```

Step 26. Update your Webex Teams (Spark) token if Lab 9 was completed more than 12 hours ago (in env_user.py)

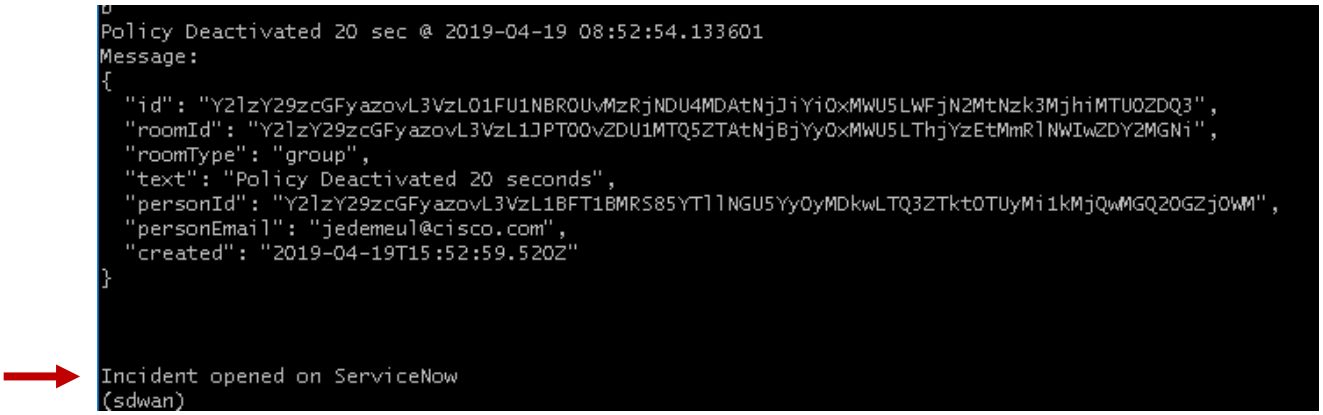
Step 27. Under the **policy_cycle(sdwan_host,session)** section at the very bottom of the bottom section of the script add the following lines:

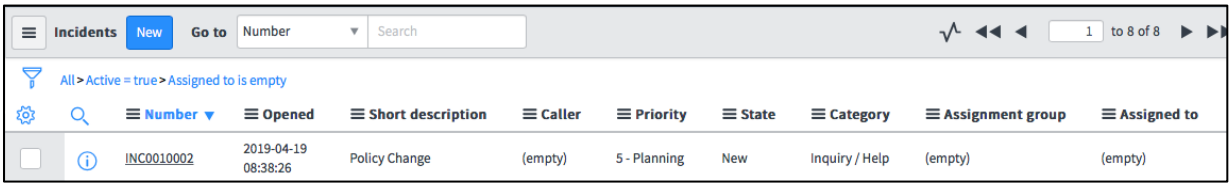
```
comments = 'There has been a policy change on : ' + sdwan_host
create_incident('Policy Change', comments, snow_user,3)
print('Incident opened on Service-Now')
```

```
session=initalize_connection(sdwan_host,sdwan_user,sdwan_pass)
if session != False:
    print ("Connection to vManage successful")
    message = spark.messages.create(env_user.SPARK_ROOM_ID,text='Python was here')
    print(message)
    print('\n\n')
    policy_cycle(sdwan_host,session)
    comments = 'There has been a policy change on : ' + sdwan_host
    create_incident('Policy Change', comments, snow_user,3)
    print('Incident opened on ServiceNow')
```

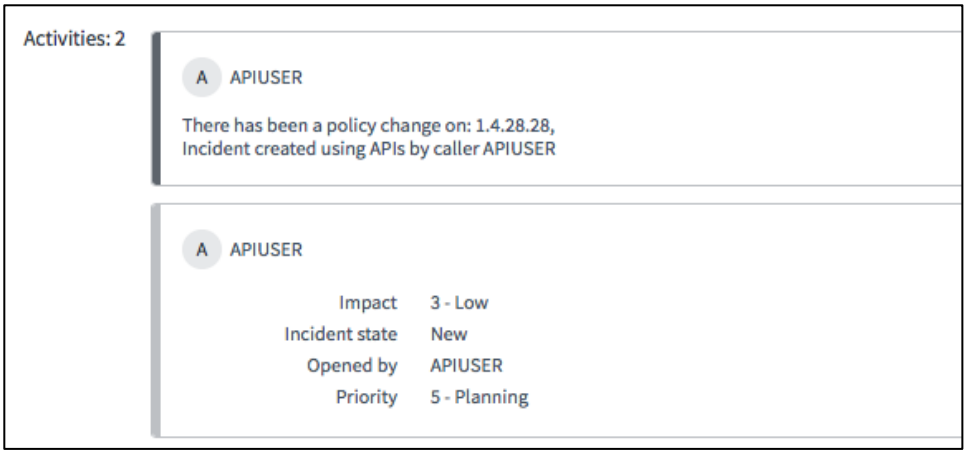
Step 28. Save your script as **PythonLab10.py**.

Step 29. Run the script using the terminal and view the output, it should post to Webex Teams and when the cycle of activate/deactivate has completed the script should create an incident in ServiceNow. Verify a new incident has been created.





Step 30. Open the incident and notice the message and priority from the script.



Step 31. **Optional Mission (Medium):** Post the ServiceNow link to the Incident to Webex Teams so it easy to navigate the incident. Required task may include:

- The ServiceNow website to see the URL structure required
- Extract the Incident number from the ServiceNow response

Step 32. End of All Labs, Congratulations