

SDWAN Programmability Lab Guide vA

June 17, 2019

Jed Demeule

Table of Contents

Table of Contents.....	2
Lab Overview	3
Lab Flow	3
Accessing the lab	3
Topology – All Labs	7
Lab 1 – Setting up the Admin PC for Development	7
Lab 2 - Exploring the REST API	10
Lab 3 - Using Postman with the REST API.....	13
Lab 4 – Installing Python and Setting up your Python Development Environment.....	17
Lab 5 - Using Python Scripting and the REST API	24
Lab 6 – Using the vManage REST API to Gather Information.....	29
Lab 7 – Using the vManage REST API to Monitor the Deployment	34
Lab 8 – Using the vManage REST API to Manage and Configure SDWAN devices	40
Case Study 1	40
Lab 9 – Posting to Webex Teams when a Policy is Activated or Deactivated	46
Case Study 2.....	46
Lab 10 – Opening a Ticket in ServiceNow.....	50
Case Study 3	50



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

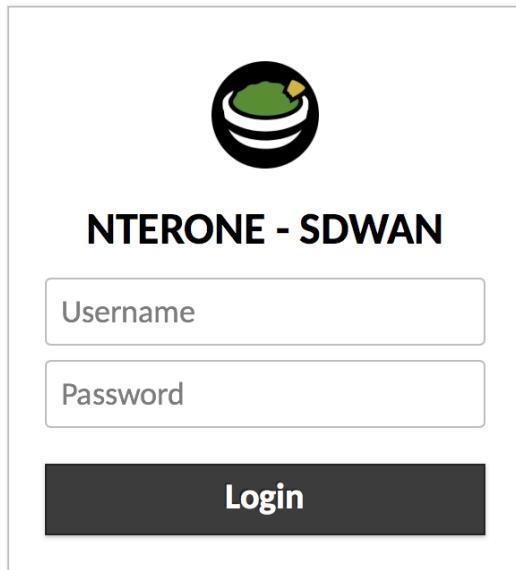
Access URL:

http://

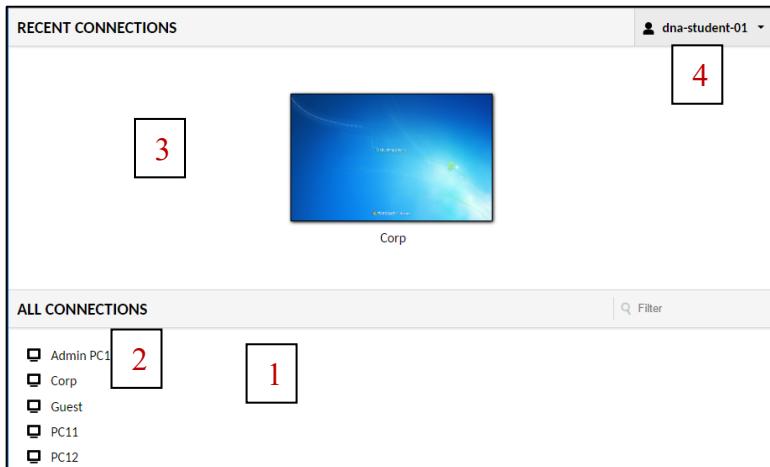
Activity Procedure

Complete these steps:

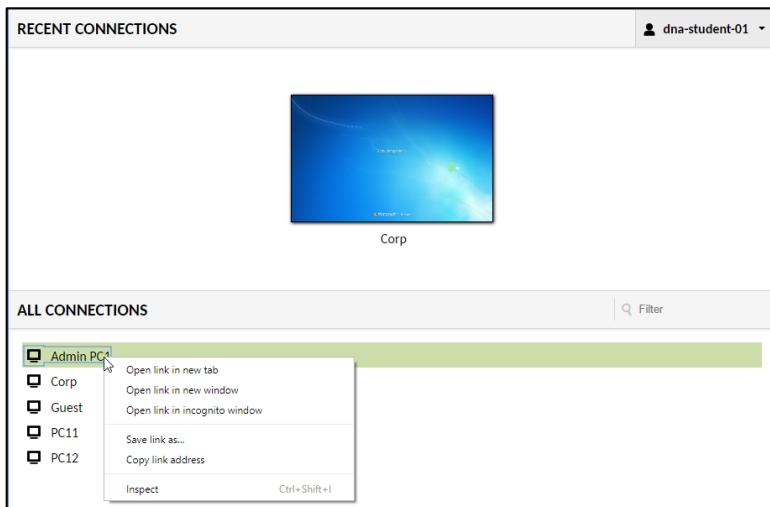
- Step 1** Use a web browser of your choice, and browse to the <http://labs.nocular.com>. You should see the login screen below:



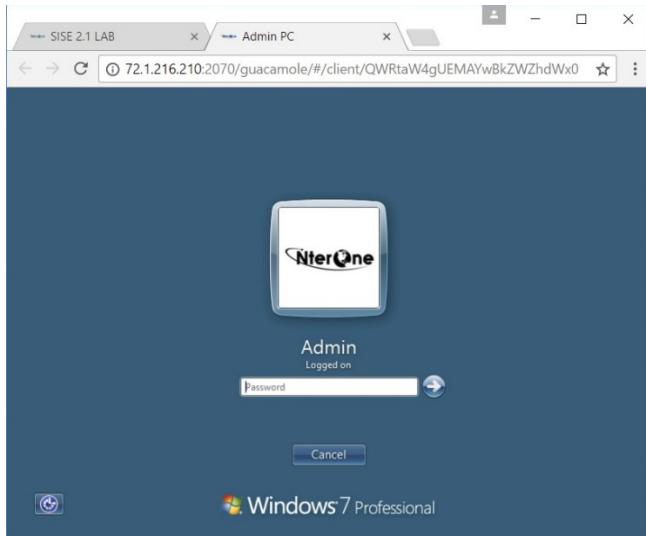
- Step 2** Enter the credentials provided by your instructor and click **Login**.
- Step 3** Observe the list of connections available (1), the type of devices (2), the Recent Connections pane (3), and the Menu Bar (4).



- 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	SSH - Management	Username	Password
vManage-1	https://1.4.28.28	10.80.88.131	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:~$
```

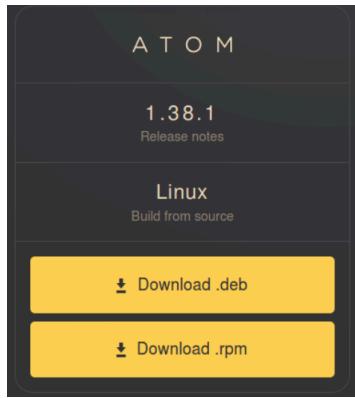
- Step 5. You should be in your home directory, you will download the class files using Git
Step 6. Attempt to clone a repository from GitHub by entering **git clone https://github.com/deplorablejed/SDWAN-Programmability**.

```
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 command **cd 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.

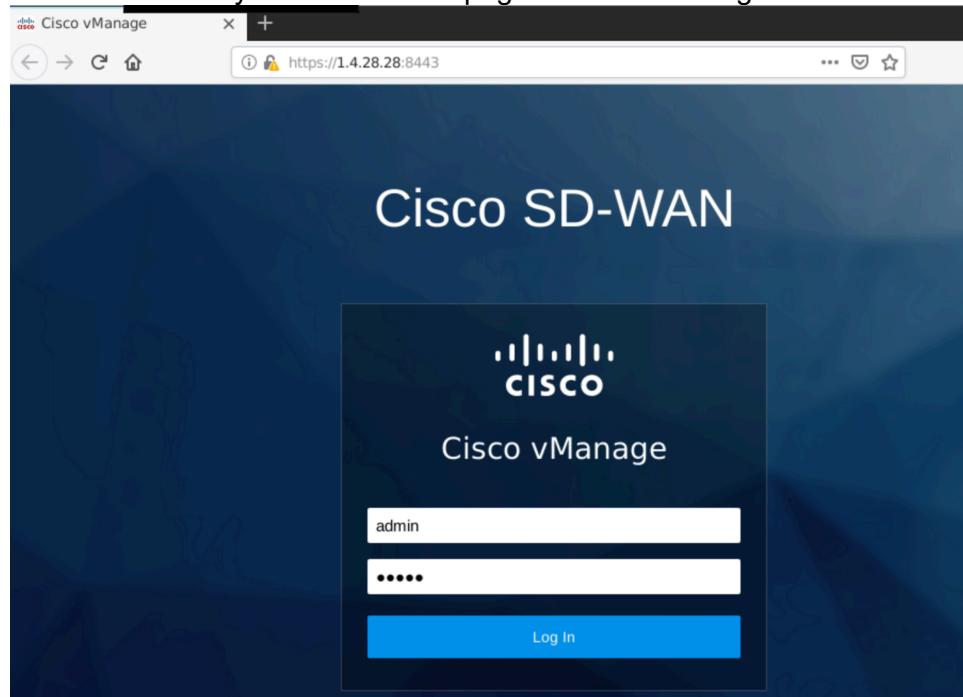
- Step 11. Provide your instructor with your email address associated with the account and ask him or her to add you to the classroom space.
-

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	SSH - Management	Username	Password
vManage-1	https://1.4.28.28	10.80.88.131	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>.



Cisco vManage API Docu X +

https://1.4.28.28/apidocs/

api_key Explore

Capacity Show/Hide | List Operations | Expand Operations | Raw

Utility - Logging Show/Hide | List Operations | Expand Operations | Raw

Alarms - Notifications Show/Hide | List Operations | Expand Operations | Raw

Diagnostics Show/Hide | List Operations | Expand Operations | Raw

CloudDock-Service Chain Show/Hide | List Operations | Expand Operations | Raw

Resource Pool Show/Hide | List Operations | Expand Operations | Raw

Configuration Database Cluster management Show/Hide | List Operations | Expand Operations | Raw

Monitoring-CloudDockCluster Show/Hide | List Operations | Expand Operations | Raw

CloudDock-Cluster Show/Hide | List Operations | Expand Operations | Raw

Administration - Tenant Show/Hide | List Operations | Expand Operations | Raw

CloudDock-Attach Show/Hide | List Operations | Expand Operations | Raw

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

Device inventory - Device		Show/Hide List Operations Expand Operations Raw
POST	/system/device/fileupload	Post form
POST	/system/device	Create device
GET	/system/device/management/systemip	Get Management system ip mapping
GET	/system/device/{deviceCategory}	Get devices details
PUT	/system/device/{uuid}	Edit device
DELETE	/system/device/{uuid}	Delete vedges
GET	/system/device/controllers/vedge/status	Get controllers vedge sync status
POST	/system/device/bootstrap/devices	Get bootstrap config for software vEdges
PUT	/system/device/decommission/{uuid}	Decommission software vEdge
GET	/system/device/bootstrap/device/{uuid}	Get bootstrap config for software vEdge
GET	/system/device/bootstrap/download/{id}	Get bootstrap config for software vEdges

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**

Parameters	
Parameter	Value
deviceCategory	vedges
model	vedge-cloud
state	

- 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.

Try it out!
[Hide Response](#)

Request URL

```
https://1.4.28.28:443/dataservice/system/device/vedges?model=vedge-cloud&&&
```

- 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.

Request URL

```
https://1.4.28.28:443/dataservice/system/device/vedges?model=vedge-cloud&&&
```

Response Body

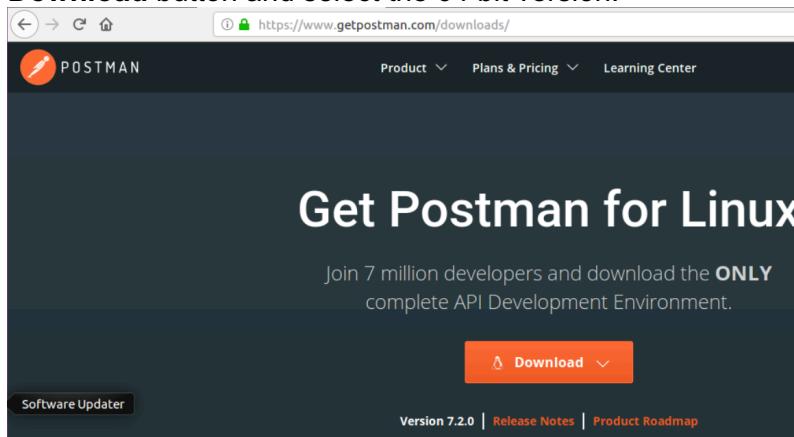
```
{
    "template": "Branch-vE2",
    "templateId": "655a459e-29d3-4c41-aefd-3d149486e8d8",
    "lifeCycleRequired": true
},
{
    "deviceType": "vedge",
    "serialNumber": "c29cc557",
    "ncsDeviceName": "vedge-7bbfd5f5-c5e3-44c6-b764-bffe982ceef5",
    "configStatusMessage": "In Sync",
    "templateApplyLog": [
        "[5-Feb-2019 15:59:49 UTC] Configuring device with cli template: DC-vE6",
        "[5-Feb-2019 15:59:49 UTC] Generating configuration from template",
        "[5-Feb-2019 15:59:49 UTC] Checking and creating device in vManage",
        "[5-Feb-2019 15:59:50 UTC] Device is online",
        "[5-Feb-2019 15:59:50 UTC] Updating device configuration in vManage",
        "[5-Feb-2019 15:59:51 UTC] Pushing configuration to device",
        "[5-Feb-2019 15:59:58 UTC] Template successfully attached to device"
    ],
    "uuid": "7bbfd5f5-c5e3-44c6-b764-bffe982ceef5",
    "lastSyncTime": "2019-02-05T15:59:58Z"
}
```

Lab 3 - Using Postman with the REST API

Device Name	vManage GUI and REST API	SSH - Management	Username	Password
vManage-1	https://1.4.28.28	10.80.88.131	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 your 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** 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 8. Notice the **Params** tab is populated with **model** and **vedge-cloud**

Params	Authorization	Headers	Body	Pre-request Script	Tests
KEY				VALUE	
<input checked="" type="checkbox"/> model				vedge-cloud	
Key				Value	

- Step 9. Next select the **Authorization** tab and select the type to **Basic Auth** and then enter a username of **admin** with a password of **admin**.

Params	Authorization	Headers	Body	Pre-request Script	Tests
Type	Basic Auth			Username	admin
				Password	admin
					<input checked="" type="checkbox"/> Show Password
The authorization header will be automatically generated when you send the request. Learn more about					

- Step 10. Press the blue **Send** button and notice there is an error.

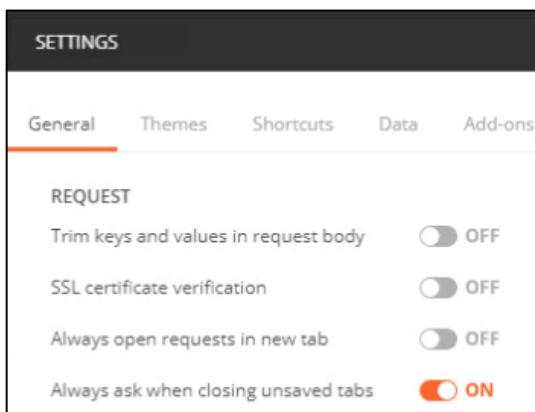
Could not get any response

There was an error connecting to <https://1.4.28.28:443/dataservice/system/device/vedges?model=vedge-cloud&&&>.

Why this might have happened:

- The server couldn't send a response: Ensure that the backend is working properly
- Self-signed SSL certificates are being blocked: Fix this by turning off 'SSL certificate verification' in *Settings > General*
- Proxy configured incorrectly: Ensure that proxy is configured correctly in *Settings > Proxy*
- Request timeout: Change request timeout in *Settings > General*

- Step 11. Follow the instructions for the second bullet point to 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 left of the interface and select **Settings**. Toggle the **SSL certificate verification** to **OFF**.



- Step 12. Close the Settings window and try to send the GET request again. This time it should be successful.

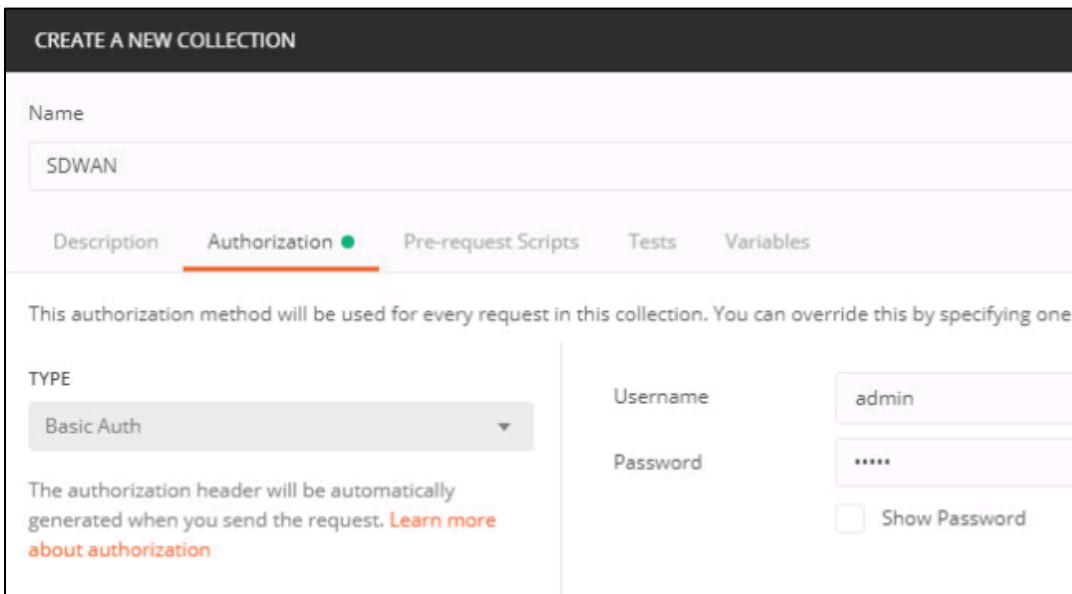
Step 13. Starting at line 207-252 take a couple minutes to look at all the returned information about the vEdge with a hostname of Branch-vE2

```

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      "chassisNumber": "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 14. Make a new collection by using the drop down triangle on the right of the orange **New** button. Give the collection a name of **SDWAN**, set the Authorization to **Basic Auth** (the username and password should be auto filled for you). Press the **Create** button when done.



CREATE A NEW COLLECTION

Name
SDWAN

Description Authorization ● Pre-request Scripts Tests Variables

This authorization method will be used for every request in this collection. You can override this by specifying one in each request.

TYPE
Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Username: admin
Password: *****
 Show Password

Step 15. You should use Postman as you are developing applications and script to test your API syntax and results.



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	SSH - Management	Username	Password
vManage-1	https://1.4.28.28	10.80.88.131	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 commands are listed below:
- **sudo apt-get install build-essential checkinstall**
 - **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**
 - **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**
 - **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. First create a Python 3 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 -m venv hello**.
- ```
ubuntu@zzDNA:~$ python3 -m venv hello
```
- Step 5. 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 6. Now in the virtual environment enter the command **python -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 7. 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 -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 8. 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 9. Start a virtual environment again based on the previous steps (you may even want to make a shell script to automate this)
- Step 10. Open a terminal run the command **pip install requests**. This module will be used in the next task.

```
admin@DESKTOP-IH4KQUS MINGW64 ~
$ pip install requests
Collecting requests
 Using cached https://files.pythonhosted.org/packages/7d/e3/20f3d364d6c8e5d2353
c72a67778eb189176f08e873c9900e10c0287b84b/requests-2.21.0-py2.py3-none-any.whl
Collecting idna<2.9,>=2.5 (from requests)
 Using cached https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1
cf41cd03869a46fe7226e7450af7a6545bfc474c9/idna-2.8-py2.py3-none-any.whl
Collecting certifi>=2017.4.17 (from requests)
 Downloading https://files.pythonhosted.org/packages/60/75/f692a584e85b7eaba0e0
3827b3d51f45f571c2e793dd731e598828d380aa/certifi-2019.3.9-py2.py3-none-any.whl (
158kB)
Collecting urllib3<1.25,>=1.21.1 (from requests)
 Using cached https://files.pythonhosted.org/packages/62/00/ee1d7de624db8ba7090
d1226aebefab96a2c71cd5cfa7629d6ad3f61b79e/urllib3-1.24.1-py2.py3-none-any.whl
Collecting chardet<3.1.0,>=3.0.2 (from requests)
 Using cached https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b64
87b4bb1ddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl
Installing collected packages: idna, certifi, urllib3, chardet, requests
Successfully installed certifi-2019.3.9 chardet-3.0.4 idna-2.8 requests-2.21.0 u
rllib3-1.24.1
You are using pip version 10.0.1, however version 19.0.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' comm
and.
(hello)
```

- Step 11. Run the command **pip install configparser**. This module will be required in the next task.

```
admin@DESKTOP-IH4KQUS MINGW64 ~
$ pip install configparser
Collecting configparser
 Downloading https://files.pythonhosted.org/packages/ba/05/6c96328e92e625fc31445d24d75a2c92ef9ba34fc5b037fe69693c362a0d/c
onfigparser-3.7.4-py2.py3-none-any.whl
Installing collected packages: configparser
Successfully installed configparser-3.7.4
You are using pip version 10.0.1, however version 19.0.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
(hello)
```

- Step 12. 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 13. Lets make sure our 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`

```
admin@DESKTOP-IH4KQU5 MINGW64 ~
$ winpty python.exe
Python 3.6.7 (v3.6.7:6ec5cf24b7, Oct 20 2018, 13:35:33) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> import urllib3
>>> import configparser
>>>
```

- Step 14. 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> + z or type `exit()` and then <Enter>.

---

- Step 15. 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
```

---

|             |                                                                                                                                                                                                                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Note</b> | 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 16. 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 17. 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 18. 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** if 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 19. From the PC2 Terminal 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 parameters:

- 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 20. Enter deactivate to tear down your virtual environment.



# 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 | SSH - Management | Username | Password |
|-------------|--------------------------|------------------|----------|----------|
| vManage-1   | https://1.4.28.28        | 10.80.88.131     | 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 IDLE use the **File → New File** menu to create a new python script  
 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`

```

1 import requests
2 import urllib3
3 import configparser

```

- Step 4. Lets make a connection to the vManage by entering the following lines (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
"""

7 def initialize_connection(ipaddress,username,password):
8
9 """
10 This function will initialize a connection to thevManage
11 :param ipaddress: This is the IP Address and Port number of vManage
12 :param username: This is the username for vManage (admin in our lab)
13 :param password: This is a password for vManage (admin in our lab)
14 These will be set in a file called package_config.ini
15 """

```

- 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)

7 def initialize_connection(ipaddress,username,password):
8
9 """
10 This function will initialize a connection to thevManage
11 :param ipaddress: This is the IP Address and Port number of vManage
12 :param username: This is the username for vManage (admin in our lab)
13 :param password: This is a password for vManage (admin in our lab)
14 These will be set in a file called package_config.ini
15 """
16
17 # Disable warnings Like unsigned certificates, etc.
18 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()

```

```

17 # Disable warnings like unsigned certificates, etc.
18 urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
19
20 url="https://"+ipaddress+"/j_security_check"
21
22 payload = "j_username="+username+"&j_password="+password
23 headers = {
24 'Content-Type': "application/x-www-form-urlencoded",
25 }
26
27 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

29 # Handle exceptions if we cannot connect to the vManage
30 try:
31 response = sess.request("POST", url, data=payload, headers=headers,verify=False,timeout=10)
32 except requests.exceptions.ConnectionError:
33 print ("Unable to Connect to "+ipaddress)
34 return False
35
36 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")
session=initialize_connection(serveraddress,username,password)
if session != False:
 print ("Successful you will issue API commands here in later labs")

```

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

- Step 11. Go to the Terminal and change to the C:\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 -m venv sdnw**.

- 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 sdnw/Scripts/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 PythonLab5.py** to execute the script you made.

```
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
```

- 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.

# 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 | SSH - Management | Username | Password |
|-------------|--------------------------|------------------|----------|----------|
| vManage-1   | https://1.4.28.28        | 10.80.88.131     | 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 default**" enter the following lines:

```

def get_inventory(serveraddress,session):
 print("Retrieving the inventory data from the vManage at "+serveraddress+"\n")
37
38 def get_inventory(serveraddress,session):
39 print("Retrieving the inventory data from the vManage at "+serveraddress+"\n")
40
41 # Open up the configuration file and get all application defaults
42 try:

```

- Step 4. Now you will define the REST API URL that will be used to fetch the inventory from 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)

38 def get_inventory(serveraddress,session):
39 print("Retrieving the inventory data from the vManage at "+serveraddress+"\n")
40
41 url = "https://" + serveraddress + "/dataservice/device"
42 response = session.request("GET",url,verify=False,timeout=10)
43 json_string = response.json()
44 print(json_string)

```

- Step 5. At the bottom of the script add an indented line  
**get\_inventory(serveraddress,session)**

```

session=initialize_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 C:\SDWAN-Programmability directory (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.

```

admin@DESKTOP-IH4KQUS 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

{'header': {'generatedOn': 1555415811333, 'viewKeys': {'uniqueKey': ['system-ip'], 'preferenceKey': 'system-ip'}, 'columns': [{"title": "Hostname", "property": "host-name", "display": "iconAndText", "iconProperty": "host-name", "hideable": False, "icon": [{"key": "vmanage", "value": "images/vmanage_table.png"}, {"key": "vedges", "value": "images/vedge_table.png"}, {"key": "vedge-vbond", "value": "images/vedge-vbond_table.png"}, {"key": "vsmart", "value": "images/vsmart_table.png"}, {"key": "vbond", "value": "images/vbond_table.png"}], "width": 150}, {"title": "State", "property": "state", "display": "iconAndToolTip", "iconProperty": "state", "toolTipDescription": "Reachability", "defaultPropertyValue": "reachable", "icon": [{"key": "green", "value": "images/device_state_green.png"}, {"key": "red", "value": "images/device_state_red.png"}, {"key": "yellow", "value": "images/device_state_yellow.png"}], "key": "default", "value": "images/device_state_default.png"}, "width": 20}, {"title": "System IP", "property": "system-ip", "hideable": False, "dataType": "string"}, {"title": "Reachability", "property": "reachability", "display": "multiColumns", "value": "616161"}, {"key": "reachable", "value": "reachable", "key": "unreachable", "value": "unreachable"}, {"key": "auth-failed", "value": "auth-failed"}, {"key": "stale", "value": "stale"}], "width": 150}], "data": [{"hostname": "1.4.28.28", "state": "reachable", "reachability": "reachable", "system_ip": "1.4.28.28", "last_update": "2018-05-15T13:33:33.000Z", "last_error": null, "stale": false}]}

```

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

```

inv={}
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-IH4KQUS 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-validity": "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", "latitude": "37.666684", "longitude": "-122.777023", "isDeviceGeoData": False, "platform": "x86_64", "uptime-date": 1555047420000, "statusOrder": 4, "device-os": "next", "validity": "valid", "state": "green", "state_description": "All daemons up", "model_sku": "None", "local-system-ip": "172.1.0.18", "total_cpu_count": "6", "testbed_mode": False, "layoutLevel": 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": "07F4E30F8E9368C48C06601C5AB51980", "certificate-validity": "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": "normal", "personality": "vsmart", "device-type": "vsmart", "timezone": "UTC", "device-groups": ["\"No groups\""], "lastupdated": 1555412683910, "domain-id": "2", "board-serial": "07F4E30F8E9368C48C06601C5AB51980", "certificate-validity": "Valid", "uuid": "6e4d0357-e883-4931-853e-25e5a85ad979", "controlConnections": "13", "device-model": "vsmart", "version": "18.4.0", "connectedvManages": ["\"172.1.0.18\""], "site-id": "1012", "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": "172.1.0.17", "total_cpu_count": "2", "testbed_mode": False, "layoutLevel": 2}
```

- 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'])
return(inv)
```

```
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': 'normal', 'personality': 'vedge', 'device-type': 'vedge', 'timeLastupdated': 1555412632485, 'bfdSessionsUp': 12, 'domain-id': '1' , 'status': 'Valid', 'max-controllers': '0', 'uuid': '276da6ac-09fe-4b55-8e0c-000000000000', 'connections': '5', 'device-model': 'vedge-cloud', 'version': '18.4.0' , 'hwVersion': '30', 'ompPeers': '2', 'latitude': '37.666684', 'longitude': '-122.333333', 'uptime-date': 1555057800000, 'statusOrder': 4, 'deviceState': 'Up', 'state_description': 'All daemons up', 'model_sku': 'None', 'location': '2', 'linux_cpu_count': '1', 'testbed_mode': False, 'layoutLevel': 1}
=====

vManage-1 69DE12CE976BC070D5184152E0298516 18.4.0 11
vSmart-1 07F4E30F8E9368C48C06601C5AB51980 18.4.0 13
vSmart-2 1AA04C7DE9392B380677AOE7D1EFB055 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 1AA04C7DE9392B380677AOE7D1EFB055 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-IH4KQUS 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.

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

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['last-updated']))

```

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 6

# 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 | SSH - Management | Username | Password |
|-------------|--------------------------|------------------|----------|----------|
| vManage-1   | https://1.4.28.28        | 10.80.88.131     | 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 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 default**", 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("-----")
 print ('{0:15} {1:20} {2} {3:36}'.format(item['system-ip'],item['hos
 def get_statistics(serveraddress,session):
 print("-----")
 print("Retrieving the statistic data from the vManage at "+serveraddress)
 print("-----")

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

- 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 exactly 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. Now comment out the section that you just copied to the clipboard as you will recreate these and use them as headers in this lab. The uncommented lines are still on 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'])

def get_statistic(serveraddress,session):
 print("-----")

```

Step 7. Paste the text that is on your clipboard (uncommented) 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 8. 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['version'],item['uuid']))

 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 9. 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 10. At the very bottom of the script add an indented line

```

get_statisticss(serveraddress,session)
session=initialize_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 11. 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-InterfaceStats'}, 'columns': [{"title": 'VPN', 'property': 'vpn-id', 'width': 50, 'dataType': 'string'}, {"title": 'Interface', 'property': 'ifname', 'width': 100, 'dataType': 'string'}, {"title": 'AF Type', 'property': 'af-type', 'width': 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', 'property': 'rx-errors', 'width': 100, 'dataType': 'string'}, {"title": 'Rx Drops', 'property': 'rx-Drops', 'width': 100, 'dataType': '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-errors', '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', 'property': 'rx-kbps', 'width': 100, 'dataType': 'string'}, {"title": 'Tx PPS', 'property': 'tx-pps', 'width': 100, 'dataType': 'string'}], 'title': 'InterfaceStats'}

```

Step 12. 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 13. 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.



```
1 for stats in json_string['data']:
2 # We only want to print the ipv4 interfaces
3 if stats['af-type']!='ipv6':
4 print(' (0:9) (1:10) (2:10d) (3:10d)').format(stats['ifname'],stats['vpn-id'],
5 rx=rx+int(stats['rx-packets'])
6 tx=tx+int(stats['tx-packets']))
7 print(' (0) (1)'.format("-----","-----"))
8 print(' (0:9) (1:10) (2:10d) (3:10d)').format(" ","Total",tx,rx))
9 print ("\\n")
```

```
#print(json_string)

for stats in json_string['data']:
 # We only want to print the ipv4 interfaces
 if stats['af-type']!='ipv6':
 print('{:<9} {:<10} {:>10d} {:>10d}'.format(stats['ifname'],stats['vpn-id'],int(stats['rx-packets']),int(stats['tx-packets'])))
 rx=rx+int(stats['rx-packets'])
 tx=tx+int(stats['tx-packets']))
 print(''':>9) ''':>10) ''':>10d) ''':>10d)'''.format("-----","-----"))
 print(''':>9) ''':>10) ''':>10d) ''':>10d)'''.format(" ", "Total",tx,rx))
print ("\n")
```

- Step 14. 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                                    |
| geo/1.30                             | 0          | 5996820  | 5991827                              |
| geo/1.40                             | 0          | 5022713  | 5001593                              |
| ge0/2                                | 1          | 7        | 3549448                              |
| eth0                                 | 512        | 25       | 31686762                             |
|                                      |            | Total    | 22992062                             |
|                                      |            | Version  | 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                                    |
| geo/1.30                             | 0          | 5991874  | 5996823                              |
| geo/1.40                             | 0          | 5001615  | 5022715                              |
| ge0/0                                | 1          | 38566    | 3510893                              |
| eth0                                 | 512        | 27       | 31686702                             |
|                                      |            | Total    | 21074434                             |
|                                      |            | Version  | 63291784                             |

- Step 15. From the SDWAN folder on your desktop open the PythonLabSource7.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 16. Paste the contents of you clipboard right above the section that starts with "# Open up the configuration file and get all application default" (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 17. At the very bottom of the script enter the following line:

```

inventory=get_inventory(serveraddress,session)
get_tunnel_statistic(serveraddress,session,systemip)
session=initialize_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)
 inventory=get_inventory(serveraddress,session)
 get_tunnel_statistic(serveraddress,session,systemip)

```

- Step 18. 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 19. End of lab

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

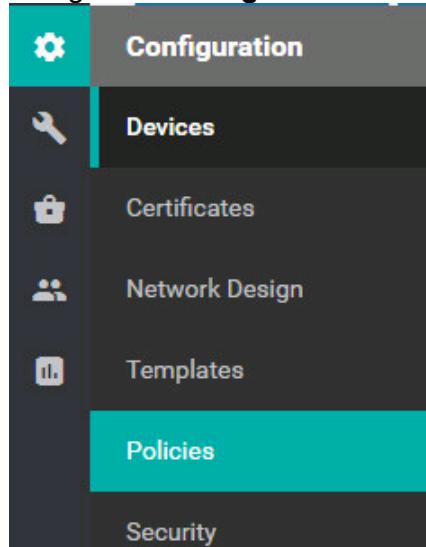
To access the vManage GUI and REST API use the http addresses listed below.

| Device Name | vManage GUI and REST API | SSH - Management | Username | Password |
|-------------|--------------------------|------------------|----------|----------|
| vManage-1   | https://1.4.28.28        | 10.80.88.131     | admin    | admin    |

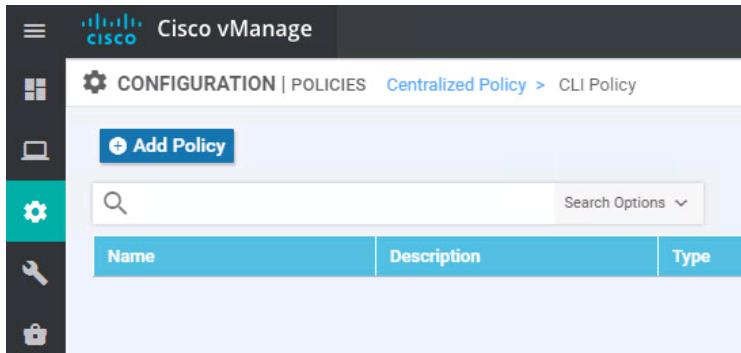
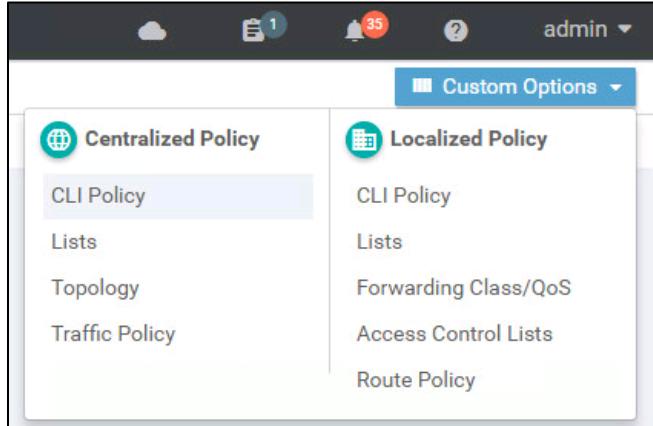
## Case Study 1

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 under **Centralized a CLI Policy**



Step 5.  
Step 6.

Press the **+Add Policy** button to add a new CLI Policy.  
Fill out the CLI Policy page with the following:

- Name: CLI\_FTP\_Policy\_PodXX
- Description: CLI FTP Policy for PodXX

| Name        | CLI_FTP_Policy |
|-------------|----------------|
| Description | CLI FTP Policy |

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
1 policy
2 data-policy _VPN1_FTP_Off_Peak
3 vpn-list VPN1
4 sequence 1
5 match
6 app-list FTP_traffic
7 source-ip 0.0.0.0/0
8 !
9 action accept
10 set
11 local-tloc-list
12 color gold
13 encaps ipsec
14 !
15 !
16 default-action accept
17 !
18 lists
19 app-list FTP_traffic
20 app ftp
21 app ftp_data
22 app ftp-data
23 !
24 site-list SiteBranch
25 site-id 10
26 site-id 20
27 site-id 30
28 !
29 site-list SiteDC
30 site-id 100
31 site-id 200
32 !
33 vpn-list VPN1
34 vpn 1
35 !
36 !
37 !
38 apply-policy
39 site-list SiteBranch
40 data-policy _VPN1_FTP_Off_Peak all
41 !
42 site-list SiteDC
43 data-policy _VPN1_FTP_Off_Peak all
44 !
45 !
46 !

```

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

| CONFIGURATION   POLICIES Centralized Policy > CLI Policy |                |      |           |
|----------------------------------------------------------|----------------|------|-----------|
| <b>Add Policy</b>                                        |                |      |           |
| <input type="text"/> Search Options ▾                    |                |      |           |
| Name                                                     | Description    | Type | Activated |
| CLI_FTP_Policy                                           | CLI FTP Policy | CLI  | 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.

| Configuration - vSmart Template Policy |                                               | Show/Hide | List Operations | Expand Operations                | Raw |
|----------------------------------------|-----------------------------------------------|-----------|-----------------|----------------------------------|-----|
| PUT                                    | /template/policy/vsmart/{policyId}            |           |                 | Edit template                    |     |
| DELETE                                 | /template/policy/vsmart/{policyId}            |           |                 | Delete template                  |     |
| POST                                   | /template/policy/vsmart                       |           |                 | Create template                  |     |
| GET                                    | /template/policy/vsmart                       |           |                 | Template list                    |     |
| GET                                    | /template/policy/vsmart/definition/{policyId} |           |                 | Get template                     |     |
| POST                                   | /template/policy/vsmart/activate/{policyId}   |           |                 | Activate vsmart policy           |     |
| POST                                   | /template/policy/vsmart/deactivate/{policyId} |           |                 | Deactivate vsmart policy         |     |
| GET                                    | /template/policy/vsmart/connectivity/status   |           |                 | Check VSmart Connectivity Status |     |

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

Configuration - vSmart Template Policy

Show

|        |                                    |
|--------|------------------------------------|
| PUT    | /template/policy/vsmart/{policyId} |
| DELETE | /template/policy/vsmart/{policyId} |
| POST   | /template/policy/vsmart            |
| GET    | /template/policy/vsmart            |

Implementation Notes  
Generate all template list

Response Class (Status )

Model Model Schema

```
{
 "header": {
 "generatedOn": 0,
 "viewKeys": "ViewKeys",
 "columns": [
 "Column"
]
 }
}
```

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.

#### Response Body

```

],
 },
 "data": [
 {
 "policyVersion": "03052019T154049661",
 "lastUpdatedBy": "admin",
 "policyName": "CLI_FTP_Policy",
 "policyDefinition": "policy\r\n data-policy _VPN1_FTP_Off_Peak\r\n vpn-list VPN1\r\n sequence 1\r\n match\r\n",
 "createdOn": 1551800449661,
 "isPolicyActivated": true,
 "policyDescription": "CLI FTP Policy",
 "@rid": 2196,
 "policyId": "c19aa591-3993-4f67-a76c-5194b368345f",
 "createdBy": "admin",
 "policyType": "cli",
 "lastUpdatedOn": 1551800449661
 }
]
}

```

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 100 seconds then will deactivate the policy for 100 seconds, this cycle will repeat 5 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.

```

1 #!/usr/bin/env python
2 """This Python Script will activate a policy for 20 seconds then deactivate the policy for 20 seconds
3 """
4
5 import requests
6 import urllib3
7 import datetime
8 import time
9 # Disable Certificate warning
10 try:
11 requests.packages.urllib3.disable_warnings()
12 except:
13 pass
14 from requests.auth import HTTPBasicAuth
15 import sys
16

```

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.

```

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

```

```

33 def policy_cycle(sdwan_host,session):
34 policy_id = "c19aa591-3993-4f67-a76c-5194b368345fE"
35 activate_policy_url = "https://%s/dataservice/template/policy/vsmart/activate/c19aa591-3993-4f67-a76c-5194b368345f"%(sdwan_host)
36 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 80 seconds for the script to finish and review the output. Notice that the policy is activated and deactivated twice.

```

$ python PythonLab8.py
SDWAN Login to 1.4.28.28 as admin ...
b''
Policy Activated 20 sec @ 2019-04-17 11:40:19.835738

b''
Policy Deactivated 20 sec @ 2019-04-17 11:40:39.843542

b''
Policy Activated 20 sec @ 2019-04-17 11:40:59.861885

b''
Policy Deactivated 20 sec @ 2019-04-17 11:41:19.869840

(hello)
admin@DESKTOP-IH4KQUS MINGW64 /c/SDWAN-Programmability (master)

```

Step 19. Take some time and look at the python scripting, notice the scripting code used is different than our previous scripts.

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:** Change the logic of the script to activate and deactivate based on a specific time of day. Share your script in the Webex Teams room setup by your instructor.

Step 23. Deactive your virtual environment when done with this lab, as you will recreate it in the next 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 | SSH - Management | Username | Password |
|-------------|--------------------------|------------------|----------|----------|
| vManage-1   | https://1.4.28.28        | 10.80.88.131     | 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 easily copy it.

---

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

---

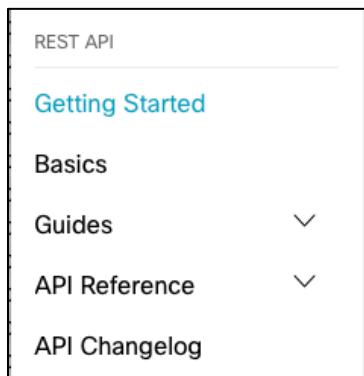
- Step 3. Copy the ID to your clipboard, this value will need to be copied into a text file found at **c:\SDWAN-Programmability\env\_user.py**. Open this file and in the copy it into the value for SPARK\_ROOM\_ID

```

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.

#### Your Personal Access Token

Bearer \*\*\*\*



- 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 = "Yjg1NWMwMjAtMTE2OC00N2RhTg0MjEtNGFjYTA4M2MzMmM4NmZlODEzNzYtZTI4_PF84_1eb65fdf-9643-417f-9974-ad72cae0e10f"
38 SPARK_ROOM_ID = "Y2lzcGJyazovL3VzL1JPT00vZDU1MTQ5ZTAtNjBjYy0xMwU5LThjYzEtMmR1NWlWZDY2MGNi"
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_usr
27 sys.path.insert(0, project_root)
28 import env_lab
29 import env_user
```

- Step 14. Below the import statements 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 20 sec@ %s "%t) and enter the following 2 lines under it (mind the indentations):

```

message = spark.messages.create(env_user.SPARK_ROOM_ID,
 text='Policy Activated 20 Sec')
 print ("Policy Activated 20 sec @ %s "%t)
 message = spark.messages.create(env_user.SPARK_ROOM_ID,
 text='Policy Activated 20 sec')
 print(message)
 print('\n\n')
 time.sleep(20)

```

- Step 16. Under the **policy\_cycle** definition find the "print ("Policy Deactivated 20 sec@ %s "%t) and enter the following 2 lines under it (mind the indentations):

```

message = spark.messages.create(env_user.SPARK_ROOM_ID,
 text='Policy Deactivated 20 Sec')
 print ("Policy Deactivated 20 sec @ %s "%t)
 message = spark.messages.create(env_user.SPARK_ROOM_ID,
 text='Policy Deactivated 20 seconds')
 print(message)
 print('\n\n')
 time.sleep(20)

```

- Step 17. Under the if session != False: (at the very bottom of the script) enter the following lines

```

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

```

- Step 18. Save your PythonLab9.py script

- Step 19. Open up a new terminal window and enter the following commands to setup a new virtual environment.

- **cd SDWAN-Programmability**
- **python3 -m venv sdwan**
- **source sdwan/bin/activate**

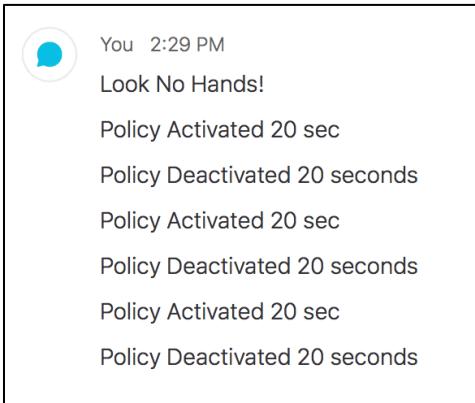
- Step 20. Open from the c:\SDWAN-Programmability folder the file called **requirements.txt**.

```

1 ciscosparkapi==0.10.1
2 requests==2.21.0
3 urllib3==1.24.1
4 xmltodict==0.11.0
5 configparser==3.7.4

```

- Step 21. Enter the command **pip install –r requirements.txt** to install all the required modules.
- Step 22. Open Webex Teams and go the Classroom space.
- Step 23. Now run the script with the command **python 3.7 PythonLab9.py** and view the output, wait for about 80 seconds for the script to finish. Verify that you see your output in the space.



You 2:29 PM  
Look No Hands!  
Policy Activated 20 sec  
Policy Deactivated 20 seconds  
Policy Activated 20 sec  
Policy Deactivated 20 seconds  
Policy Activated 20 sec  
Policy Deactivated 20 seconds

- Step 24. **Optional Mission:** 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 25. End of Lab

# Lab 10 – Opening a Ticket in ServiceNow

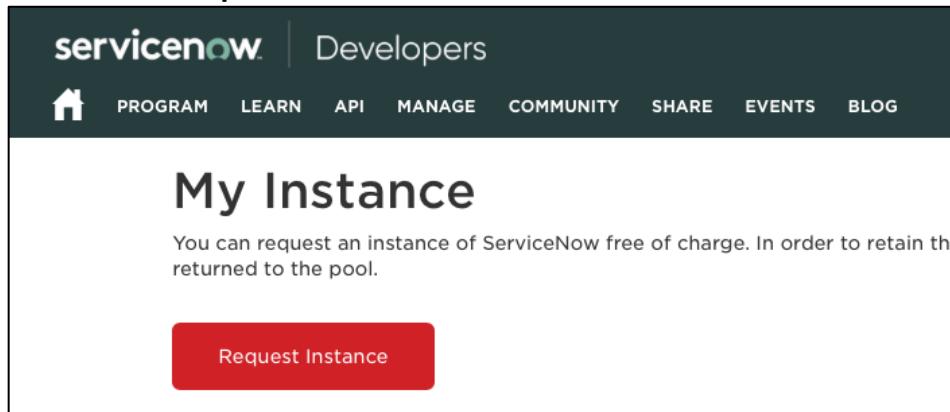
To access the vManage GUI and REST API use the http addresses listed below.

| Device Name | vManage GUI and REST API | SSH - Management | Username | Password |
|-------------|--------------------------|------------------|----------|----------|
| vManage-1   | https://1.4.28.28        | 10.80.88.131     | 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. Go to <https://developer.servicenow.com> and create an account (click Register)!
- Step 2. After activating your account, login into the developer site.
- Step 3. Create Personal Developer Instance
- Step 4. 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



The screenshot shows the ServiceNow Developers website. The top navigation bar includes links for PROGRAM, LEARN, API, MANAGE, COMMUNITY, SHARE, EVENTS, and BLOG. The main content area is titled "My Instance" and contains the text: "You can request an instance of ServiceNow free of charge. In order to retain the instance, you must agree to the terms and conditions and return it to the pool." Below this text is a large red button labeled "Request Instance".

- Step 5. Enter "**testing**" and then the **I understand** button.

**Please Note**

Your personal developer instance will be reclaimed after 10 days of inactivity. To understand how we track activity and other important information on your personal developer instance, please see the [FAQ](#).

**Please tell us how you will use your ServiceNow personal developer instance? \***

testing

7993 characters left

I understand

- Step 6. Choose the instance version of Kingston (Madrid will not work with the script we will write).
- Step 7. 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
- Step 8. Open Notepad++ and copy the URL, username and password that you copied to your clipboard.
- Step 9. Click the link for your instance and use the username of admin and the password that you copied to notepad++
- Step 10. Change your admin password to C1sc0123

Change Password

User name:  
admin

Current Password:  
\*\*\*\*\*

New password:  
\*\*\*\*\*

Confirm New Password:  
\*\*\*\*\*

**Submit**

- Step 11. 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](#).

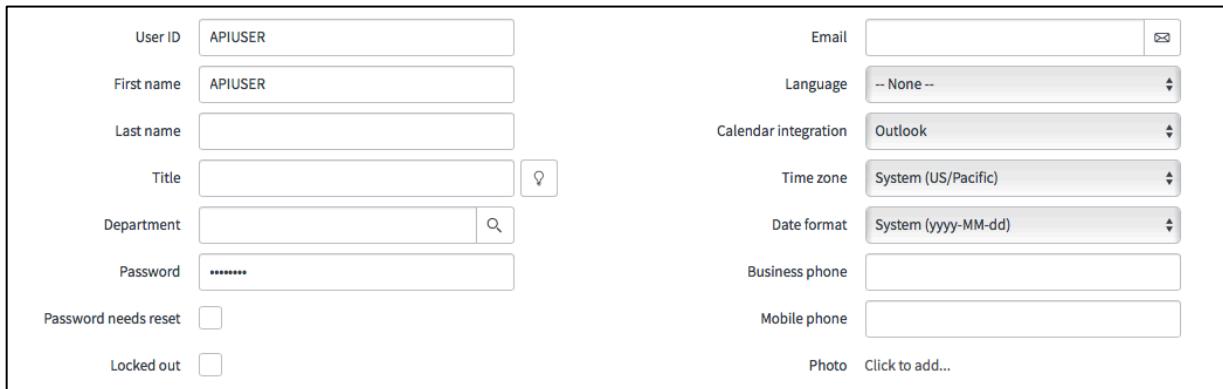
---

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

---

Step 12. Your Home page in ServiceNow should load

Step 13. Add a new user named **APIUSER** with a password of **C!sc0123** (that is a zero).

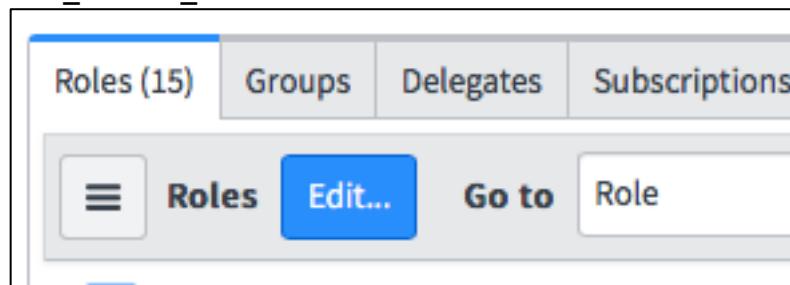


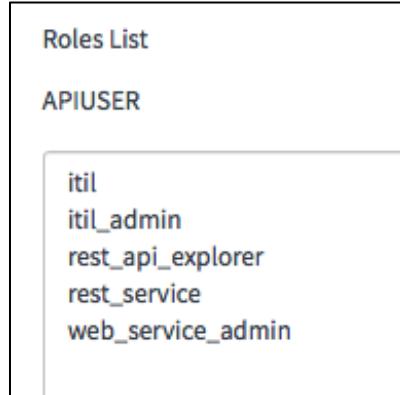
A screenshot of the ServiceNow user creation form. The fields filled in are:

- User ID: APIUSER
- Email: (empty)
- First name: APIUSER
- Language: -- None --
- Last name: (empty)
- Calendar integration: Outlook
- Title: (empty)
- Time zone: System (US/Pacific)
- Department: (empty)
- Date format: System (yyyy-MM-dd)
- Password: (redacted)
- Business phone: (empty)
- Business phone needs reset: (unchecked)
- Date format: System (yyyy-MM-dd)
- Mobile phone: (empty)
- Mobile phone needs reset: (unchecked)
- Locked out: (unchecked)
- Photo: Click to add...

Step 14. 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 15. On the left hand side navigate to **Incident → Open**  
 Step 16. In the SDWAN-Programmablity folder on the root of the C: drive open up the **env\_lab.py** script in Atom.

```
38 vManage = {
39 "host": "1.4.28.28",
40 "username": "admin",
41 "password": "admin"
42 }
```

- Step 17. 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": "C1sc0123"
}

vManage = {
 "host": "1.4.28.28",
 "username": "admin",
 "password": "admin"
}
serviceNow = {
 "url": "https://dev51787.service-now.com/api/now",
 "username": "APIUSER",
 "password": "C1sc0123"
}
```

---

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

---



- Step 18. Save the **env\_lab.py** file.
- Step 19. 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 been changed and a message written to Webex Teams.
- Step 20. Do a Save As of the script and name it **PythonLab10.py**, this will be the script you work on for this lab.
- Step 21. Add the following section under the sdwan credentials:

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

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_url = env_lab.serviceNow['url']
snow_user = env_lab.serviceNow['username']
snow_pass = env_lab.serviceNow['password']
```

- Step 22. In Atom open up PythonLab10Source in another tab and copy the entire contents to the clipboard.

```
1 def get_user_sys_id(username):
2
3 # find the ServiceNow user_id for the specified user
4
5 url = snow_url + '/table/sys_user?sysparm_limit=1&name=' + username
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, username, 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(username)
17 print caller_sys_id
18 url = snow_url + '/table/incident'
19 payload = {'short_description': description,
20 'comments': (comment + ', \nIncident created using APIs by caller ' + username),
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']
```

- Step 23. Paste the entire contents of the clipboard below the def policy\_cycle section and before the session=initialize connection

```

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=' + username
 headers = {'Content-Type': 'application/json', 'Accept': 'application/json'}
 response = requests.get(url, auth=(snow_user, snow_pass), headers=headers)
 user_json = response.json()
 return user_json['result'][0]['sys_id']

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

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

 caller_sys_id = get_user_sys_id(username)
 print caller_sys_id
 url = snow_url + '/table/incident'
 payload = {'short_description': description,
 'comments': (comment + ', \nIncident created using APIs by caller ' + username),
 '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=initialize_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='Look No Hands!')
 print(message)
 print("\n\n")
 policy_cycle(sdwan_host,session)

```

Activ  
Go to

- Step 24. Update your Webex Teams (Spark) token if Lab 9 was completed more than 12 hours ago (in env\_user.py)
- Step 25. Under the "policy\_cycle(sdwan\_host,session)" at the bottom 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 ServiceNow')

```
- Step 26. Save your script as PythonLab10.py.
- Step 27. 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.

```

Policy Deactivated 20 sec @ 2019-04-19 08:52:54.133601
Message:
{
  "id": "Y2lzY29zcGFyazovL3VzL01FU1NBR0UvMzRjNDU4MDAtNjJiYi0xMWU5LWFjN2MtNzk3MjhMTUQZDQ3",
  "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vZDU1MTQ5ZTAtNjBjYy0xMWU5LThjYzEtMmR1NWIwZDY2MGNi",
  "roomType": "group",
  "text": "Policy Deactivated 20 seconds",
  "personId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRS85YT11NGU5Yy0yMDkwLTQ3ZTk0TUyMi1kMjQwMGQ20GZj0WM",
  "personEmail": "jedemeul@cisco.com",
  "created": "2019-04-19T15:52:59.520Z"
}

```

Incident opened on ServiceNow
(sdwan)

Incidents							
	New	Go to	Number	Search			
<input type="checkbox"/> All > Active = true > Assigned to is empty							
			Number				
<input type="checkbox"/>	<input type="checkbox"/>  INC0010002	2019-04-19 08:38:26	Policy Change	(empty)	5 - Planning	New	Inquiry / Help
							(empty)
							(empty)

Step 28. Open the incident and notice the message and priority from the script.

Activities: 2

 APIUSER

There has been a policy change on: 1.4.28.28,
Incident created using APIs by caller APIUSER

 APIUSER

Impact	3 - Low
Incident state	New
Opened by	APIUSER
Priority	5 - Planning

Step 29. End of Lab