**ADVPRG**

# Advanced Programming Blitz

Student Guide

# Table of Contents

# Lab 1: Verify the Student Desktop is setup for Development

## Job Aid

The Instructor will assign you to one of the available Student Desktops. You may be required to work in pairs based on the number of attendees.

**Step 1** If you do not already have a Webex Teams account (http//teams.webex.com), set a free account up now on your personal computer. You may choose to use the web browser-based interface or install the application on your computer. The Student PC remote desktop is Ubuntu Linux and can only use the browser based interface of Webex Teams.

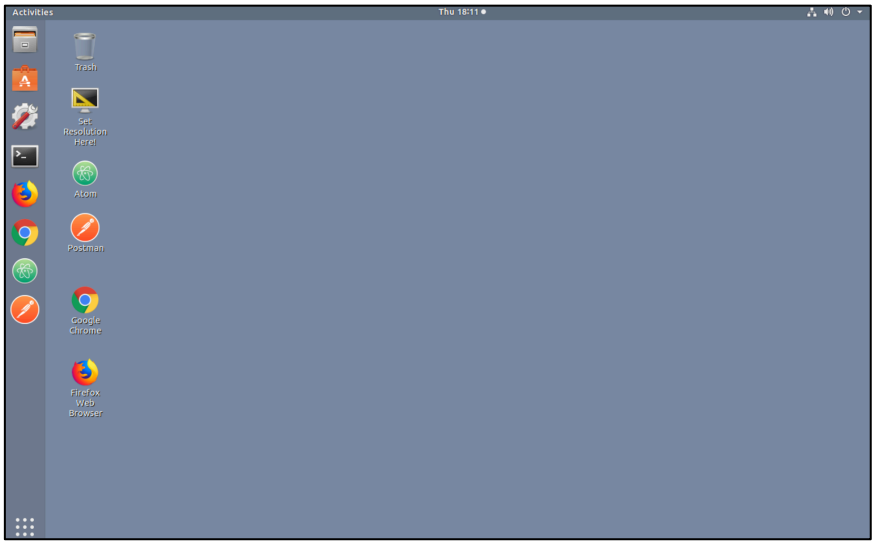
| **Note** | Webex teams will be used in the labs and also by the instructor to distribute information and files. |
|---|---|

**Step 2** Provide your instructor with your email address associated with the account and ask him or her to add you to the classroom space.

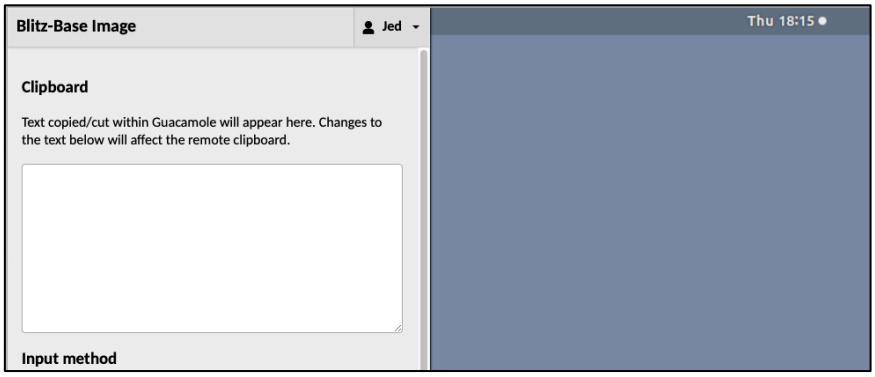
**Step 3** Your instructor will provide lab credentials in the classroom space

**Step 4** Open a web browser from your computer to http://labs.nocular.com and enter the credentials assigned to you by your instructor.

**Step 5** Connect to the Student Desktop




**Step 6** Use the sidebar to allow access to cut and paste from your computer to the remote desktop by enter in the key sequence CTRL-ALT-SHIFT on Windows or COMMAND-CTRL-SHIFT on a Mac. The same sequence will close the sidebar




**Step 7** From the Student desktop open a terminal

**Step 8** Enter the command git `--version`. You should get output indicating the version of git installed.

```
student@Student-Desktop:~$ git --version
git version 2.17.1
```

**Step 9** You should be in your home directory for student; you will download the class files using the Git protocol

**Step 10** Attempt to clone a repository from GitHub by entering **git clone https://github.com/deplorablejed/Adv-Prog-Blitz**.

**Step 11** View the contents of what was just downloaded by entering the command **cd Adv-Prog-Blitz | more .** These files will be required in later labs.

**Step 12** The lab guide will assume that the text editor Atom will be used, if you like a different text editor, visual studio, PyCharm and Sublime text are installed.

# Lab 2: NETCONF and YANG

Now we shall take a look at YANG in action. In particular, let's start to connect YANG and NETCONF, so we can understand how the two relate. In order to achieve that, we need a Network Device with YANG and NETCONF Capabilities.

To start, we shall run a basic example using a simple Python script to connect to the CAT9000 device using NETCONF. After connecting to your device, the Python script prints out the NETCONF capabilities supported by the CAT9000 (acting as a NETCONF server). Use the following IP Address 10.10.10.XX (your instructor will provide the last octet)and establish a SSH connection and login with the following credentials: admin/ISEisC00L

To access the CAT9000 use the addresses listed below.

| Device Name | IP Address | Port | Username | Password |
|---|---|---|---|---|
| CAT9000 | 10.10.10.XXX (ask instructor) | 830 | admin | ISEisC00L |

## Advanced Path (Recommended): Task 1 Configuring NETCONF and Exchanging Capabilities

**Step 1** Login the Student Desktop using http://labs.nocular.com with credentials provided by the instructor and then login to Windows 10 with **admin/APIisthenewCLI.**

**Step 2** Configure the CAT9000 to support NETCONF/YANG by doing the following from configuration mode:

- Enable **netconf-yang**
- Enable **aaa new-model**

- Enable **aaa authorization exec default local**
- Configure **username cisco privilege 15 password cisco**

**Step 3** Verify that NETCONF/YANG is configured properly with the command show platform software yang-management process

**Step 4** Open an SSH connection on port 830 from your terminal to see a list of capabilities. Use credentials of admin/ISEisC00L (those are zero's)

**Step 5** Install python3-venv and create a virtual environment called NETCONF. Then activate the virtual environment and install ncclient with pip.

**Step 6** Next you will modify a script to get the capabilities of the CAT9000. Open the get_cap.py script as the starting point and configure the script to connect to the CAT9000 device using NETCONF (this uses ncclient). The script should be run in the blitz virtual environment and will have the following properties:

- Define variables for HOST, PORT, USER and PASS
- Uses the ncclient manager to connect to the CAT9000
- Prints the capabilities to the terminal

**Step 7** The information the script returns about the CAT9000 device should be similar to the following:

```
/home/ubuntu/PycharmProjects/Class/venv/bin/python /home/ubuntu/Documents/get_cap.py
3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609]
***Here are the Remote Devices Capabilities***
http://cisco.com/ns/yang/Cisco-IOS-XE-multicast?module=Cisco-IOS-XE-multicast&revision=2017-06-01
http://cisco.com/ns/yang/Cisco-IOS-XE-aaa?module=Cisco-IOS-XE-aaa&revision=2017-09-05
urn:ietf:params:xml:ns:yang:smiv2:DISMAN-EVENT-MIB?module=DISMAN-EVENT-MIB&revision=2000-10-16
```

**Optional Bonus Mission** (Difficulty Level - Low)

- Have the script prompt you for the IP address, port, username and password.

**Optional Bonus Mission** (Difficulty Level – Medium)

- Modify the script to only return the capabilities regarding BGP

# Foundation Path: Task 1 Configuring NETCONF and Exchanging Capabilities

**Step 1** Login the Student Desktop using http://labs.nocular.com with credentials provided by the instructor**.**

**Step 2** Open the Ubuntu Desktop (note the credentials are admin/cisco)

**Step 3** From the Student Desktop desktop open Terminal and and open an SSH connection to the CAT9000 device. Use the credentials of **admin/ISEisC00L** (those are zeros)

**Step 4** Enter the following commands at the CLI to enable NETCONF/YANG, in order to be able to run correctly the Python Script:

```
configure terminal
     netconf-yang
     aaa new-model
     aaa authorization exec default local
     username cisco privilege 15 password cisco
```

**Step 5** Verify NETCONF/YANG on the CAT9000. This may take a couple of minutes before you see the services running.

**Step 6** From the Command Line Interface (CLI) of the Catalyst, this command can be used to ensure that the software processes required to support the Data Model Interface (DMI) on the Catalyst 9000 are running:

```
show platform software yang-management process
```

```
POD1-SW1-EDGE#show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Running
dmiauthd         : Running
vtyserverutild   : Running
opdatamgrd       : Running
nginx            : Running
ndbmand          : Running
pubd             : Running
```

**Step 7** Open a ssh connection on port 830 to the CAT9000 device from a terminal on your Student Desktop. Notice the switch returns its capabilities as a sort of hello.

**Step 8** Then in your preferred text editor (Atom, Pycharm, Sublime Text) select **File > New** to start a new script

**Step 9** Enter the following 3 lines to import the required **ncclient** and **sys** modules:

```
# Lab 2 Script get_cap.py
from ncclient import manager
import sys
```

The above lines will install the ncclient library and sys library. These libraries will be installed with **pip install ncclient** command in a later step.

**Step 10** Enter the following 4 lines below the previous lines, ask your instructor for the last octet of the host. This section is used to set the IP, port and credentials used to login to the CAT9000.

```
HOST = '10.10.10.XXX'
PORT = 803
```

```
USER = 'cisco'
PASS = 'cisco'
```

The above lines create local variables that we can update as needed to fit the appropriate environment. In our lab we are connecting to a CAT9000

| **Note** | A note about security: You will notice that in the examples of this learning lab, we are putting our username/password directly in the Python script rather than using public keys this is not advised in production environment. |
|---|---|

**Step 11** Add the following lines below the previous 4 lines:

```
def main():
    """
    Main method that prints netconf capabilities of remote device.
    """
    with manager.connect(host=HOST, port=PORT, username=cisco,
                         password=cisco, hostkey_verify=False,
                         device_params={'name': 'default'},
                         look_for_keys=False, allow_agent=False) as m:

        # print all NETCONF capabilities
        print('***Here are the Remote Devices Capabilities***')
        for capability in m.server_capabilities:
            print(capability)
```

**Step 12** Let's look at what the code above is doing:

**Step 13** The with ... as expression illustrated below creates the NETCONF over SSH session with the following required arguments:

- `host` = the IP address or hostname of the remote device
- `port` = the NETCONF port for the SSH session
- `username` = the username to authenticate the SSH session
- `password` = the password to authenticate the SSH session
- `hostkey_verify` = disables hostkey verification from ~/.ssh/known_hosts
- `device_params` = allows for vendor specific operations (in this case for any NETCONF device)
- `look_for_keys` = disables public key authentication since we are using username/password
- `allow_agent` = disables public key authentication since we are using username/password

Next, once we have established our SSH session, we have a for loop which iterates over a list called `m.server_capabilities` which contains the NETCONF capabilities advertised by the NETCONF server.

Also, you will notice that the examples in this learning lab use `hostkey_verify=False` to ignore your `known_hosts` file.

Next, once we have established our SSH session, we have a `for` loop which iterates over a list called `m.server_capabilities` which contains the NETCONF capabilities advertised by the NETCONF server.

Also, you will notice that the examples in this learning lab use the `hostkey_verify=False` to ignore your `known_hosts` file.

**Step 14** Finally lets add code at the bottom ensures that our **main()** method is only executed when the script is called directly (instead of being imported as a module) by entering the following lines:

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

**Step 15** 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 16** Make sure you are in your home directory (cd /home/ubuntu). Failure to do so will result in permission errors when you try and run the following commands.

**Step 17** First create a Python 3.7 virtual environment using the **venv** module included with Python 3. The virtual environment can be anything that you want, but in this lab lets call it **NETCONF**. Enter the command **python3.7 -m venv NETCONF**.

**Step 18** Now "activate" the environment. Look for the name of the virtual environment to be enclosed in parenthesis after activation. Enter the command

**source blitz/bin/activate**.

**Step 19** Now in the virtual environment enter the command **python3.7 –V** and verify Python 3.7.3 is active in this virtual environment.

**Step 20** Enter the command **pip install ncclient** to install the required module

```
(YANG) c:\>pip install ncclient
Collecting ncclient
  Downloading https://files.pythonhosted.org/packages/36/b1/1f909193588dfd35726e8ae35e01463386bbcf670b90ed61c4930ce447db/ncclient-0.6.6.tar.gz (89kB)
    |████████████████████████████████| 92kB 2.0MB/s
Requirement already satisfied: setuptools>0.6 in c:\yang\lib\site-packages (from ncclient) (41.0.1)
Collecting paramiko>=1.15.0 (from ncclient)
  Downloading https://files.pythonhosted.org/packages/4b/80/74dace9e48b0ef923633dfb5e48798f58a168e4734bca8ecfaf839ba051a/paramiko-2.6.0-py2.py3-none-any.whl (199kB)
    |████████████████████████████████| 204kB 6.8MB/s
Collecting lxml>=3.3.0 (from ncclient)
  Downloading https://files.pythonhosted.org/packages/90/e2/fa3d40b4de170d35c3f6b4cc4c25ef0eaea93d08f598a4d7c0d61bd2886b/lxml-4.4.0-cp36-cp36m-win_amd64.whl (3.7MB)
    |████████████████████████████████| 3.7MB 4.7kB/s
Collecting six (from ncclient)
  Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.whl
```

**Step 21** Using an SSH connection to the CAT9000 on port 830.

**Step 22** When prompted for credentials use **cisco/cisco** and notice that the capabilities are sent as a form of hello. Notice this list is much longer that the screenshot.

```
login as: cisco
cisco@10.10.10.14's password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:base:1.1</capability>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability
```

**Step 23** From the NETCONF virtual environment run the script by entering the command **python get_cap.py** and verify that the output is similar to the following (the screenshot has been truncated):

```
(YANG) c:\Adv-Prog-Blitz>python get_cap.py
***Hear are the Remote Devices Capabilities***
urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:xpath:1.0
urn:ietf:params:netconf:capability:validate:1.0
urn:ietf:params:netconf:capability:validate:1.1
urn:ietf:params:netconf:capability:rollback-on-error:1.0
urn:ietf:params:netconf:capability:notification:1.0
urn:ietf:params:netconf:capability:interleave:1.0
```

## Optional Bonus Mission (Difficulty Level - Low)

- Have the script prompt you for the IP address, port, username and password.

## Optional Bonus Mission (Difficulty Level – Medium)

- Modify the script to only return the capabilities regarding BGP

**Step 24** **End of Lab**

# Advanced Path (Recommended): Task 2: Using NETCONF/YANG to Gather Information

**Step 1** Enable RESTCONF and NETCONF

**Step 2** Examine the netconf_restconf.py script

**Step 3** Examine the utils.py script

**Step 4** Modify the Python script called config.py in the root of the BlitzAdvanced-master folder so that it defines the following variables and the appropriate values (10.10.10.XXX, admin, ISEisC00L, 830)

- IOS_XE_HOST
- IOS_XE_USER
- IOS_XE_PASS
- IOS_XE_PORT

**Step 5** Create a new script called myxe.py, in the script import the following modules

- requests
- urllib3
- ncclient
- xml
- xml.dom.minidom
- json
- sys

**Step 6** Add to the myxe.py the following functions

- import utils, and netconf_restconf
- allow self-signed certificates
- get the variables from the config.py script
- fetch the hostname of the device and display on the screen

**Step 7** Create a virtual environment called XE and run your myxe.py script. Make sure the appropriate packages are installed!

## Optional Bonus Mission (Difficulty Level - low)

- Use a NETCONF function to return an interfaces operational data and output it.

# Foundation Path: Task 2: Using NETCONF/YANG to Gather Information

**Step 1** Open a terminal and use

**Step 2** Enter the command **ip http secure-server**

**Step 3** Enter the command **restconf**

**Step 4** Enter the command **netconf-yang**

**Step 5** Open the file netconf_restconf.py located in the "ios_xe_programmability" folder within the "BlitzAdvanced-master" folder

**Step 6** Review some of the functions the file has:

```
def get_netconf_hostname(ios_xe_host, ios_xe_port, ios_xe_user, ios_xe_pass):
    """
    This function will retrieve the device hostname via NETCONF
    :param ios_xe_host: device IPv4 address
    :param ios_xe_port: NETCONF port
    :param ios_xe_user: username
    :param ios_xe_pass: password
    :return: IOS XE device hostname
    """
```

```
def get_restconf_hostname(ios_xe_host, ios_xe_user, ios_xe_pass):
    """
    This function will retrieve the device hostname via RESTCONF
    :param ios_xe_host: device IPv4 address
    :param ios_xe_user: username
    :param ios_xe_pass: password
    :return: IOS XE device hostname
    """
```

```
def get_netconf_int_oper_data(interface, ios_xe_host, ios_xe_port, ios_xe_user, ios_xe_pass):
    """
    This function will retrieve the operational data for the interface via NETCONF
    :param interface: interface name
    :param ios_xe_host: device IPv4 address
    :param ios_xe_port: NETCONF port
    :param ios_xe_user: username
    :param ios_xe_pass: password
    :return: interface operational data in XML
    """
```

```
def get_restconf_int_oper_data(interface, ios_xe_host, ios_xe_user, ios_xe_pass):
    """
    This function will retrieve the operational data for the interface via RESTCONF
    :param interface: interface name
    :param ios_xe_host: device IPv4 address
    :param ios_xe_user: username
    :param ios_xe_pass: password
    :return: interface operational data in JSON
    """
```

| **Note** | Notice that there are RESTCONF and NETCONF version of each functions. |
|---|---|

**Step 7** Edit the **config.py** file (save when done) that you can find in the "BlitzAdvanced-master" folder and enter your values for (double check with your instructor the IP Address of your IOS_XE Device):

```
IOS_XE_HOST = '10.10.10.XXX'
IOS_XE_USER = 'admin'
IOS_XE_PASS = 'ISEisC00L'
IOS_XE_PORT = '830'
```

**Step 8** Create a new Python file called **myxe.py** (use Atom) and write down or copy/paste the lines below into the beginning of the file:

```
#Lab 3a
import requests
import urllib3
import ncclient
import xml
import xml.dom.minidom
import json
import sys

sys.path.append('..')

import utils
import netconf_restconf


from ncclient import manager
from urllib3.exceptions import InsecureRequestWarning
from requests.auth import HTTPBasicAuth  # for Basic Auth
from config import IOS_XE_HOST, IOS_XE_PORT, IOS_XE_USER, IOS_XE_PASS

urllib3.disable_warnings(InsecureRequestWarning)  # Disable insecure https
      warnings
```

**Step 9** Add to this file, after all the lines you've written or copy/pasted the correct function to display the Device's Hostname. Use the file netconf_restconf.py (located in the ios_xe_programmability folder) to look for the right NETCONF function definition.

Repeat step 6 but now to

# Lab 3: WebEx Teams API

Webex Teams

## Introduction

In this Lab we will make REST API calls to GET (read), POST (create), PUT (modify), DEL (delete), Webex Teams Rooms, Messages, URLs and Files using Python Programming. We will then use Webex Teams Libraries to simplify the process.

## Activity

In this activity, you will investigate: Review REST API Components and the WebEx Teams Interactive DOCs.

## Required Resources

These are the resources and equipment that are required to complete this activity:

Student Desktop, Internet Access and a Webex Teams Account (You could also use your own Laptop as long as you have Python installed, the Virtual Environment activated, the required libraries specified in the **Requirements.txt** file installed and the appropriate git repository cloned).

# Common Path for Advanced and Foundation – Task 1: Build GET/POST calls using Python to automate Rooms and Message Creation

## Required Resources

These are the resources and equipment that are required to complete this activity:

Student Desktop, Internet Access a Webex Teams Account (you can get one for free) and the files contained in the "*BlitzAdvanced-master*" folder. (You could also use your own Laptop as long as you have Python installed, the Virtual Environment activated, the required libraries specified in the **Requirements.txt** file installed and the folder "*BlitzAdvanced-master*" with all its content available).

## Python Requests Library

These Code Samples use the Python Requests Library to simplify making REST API calls.

## Webex Teams Account and Developer Access Token

You will need a Webex Teams Account to complete this lab. If you do not have an account yet, go to Webex Teams for Developers located at https://developer.webex.com/ page, click the Sign Up button and create a free account.

# Task 2: Install the Needed Modules and Configure Path

## Activity

In order to be able to interact with the Webex Teams API, we need Libraries that provide the necessary functions to use URLs, perform requests, read/parse JSON files within the Python Script we will be running.

**Step 1** Install the following packages from the command prompt: urllib3, requests and json. (Internet Access is mandatory to complete this task)

**Step 2** Open a command prompt. Execute the commands below:

- pip install urllib3
- pip install requests
- pip install requests-toolbelt

Output should be similar to this:

```
C:\Users\Admin>pip install urllib3
Collecting urllib3
  Downloading https://files.pythonhosted.org/packages/8c/4b/5cbc4cb46095f369117d
cb751821e1bef9dd86a07c968d8757e9204c324c/urllib3-1.24-py2.py3-none-any.whl (117k
B)
    100% |████████████████████████████████| 122kB 3.3MB/s
Installing collected packages: urllib3
Successfully installed urllib3-1.24
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' comm
and.

C:\Users\Admin>pip install requests
Collecting requests
  Downloading https://files.pythonhosted.org/packages/f1/ca/10332a30cb25b627192b
4ea272c351bce3ca1091e541245cccbace6051d8/requests-2.20.0-py2.py3-none-any.whl (6
0kB)
    100% |████████████████████████████████| 61kB 2.0MB/s
Collecting chardet<3.1.0,>=3.0.2 (from requests)
  Downloading https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b648
7b4bb1ddec7ca55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl (133
kB)
    100% |████████████████████████████████| 143kB 3.3MB/s
Collecting certifi>=2017.4.17 (from requests)
  Downloading https://files.pythonhosted.org/packages/56/9d/1d02dd80bc4cd955f989
```

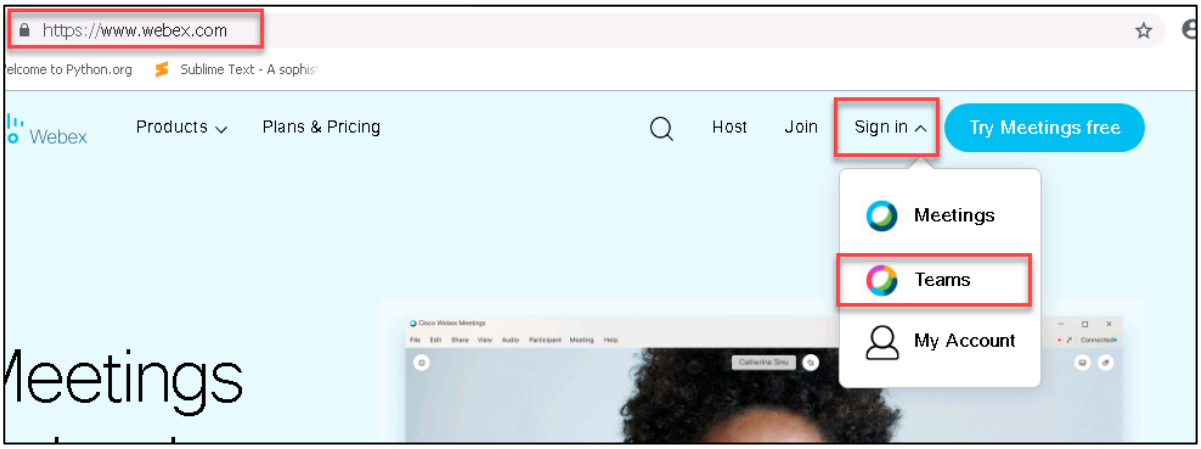
# Task 3: Log into WebEx Teams

## Activity

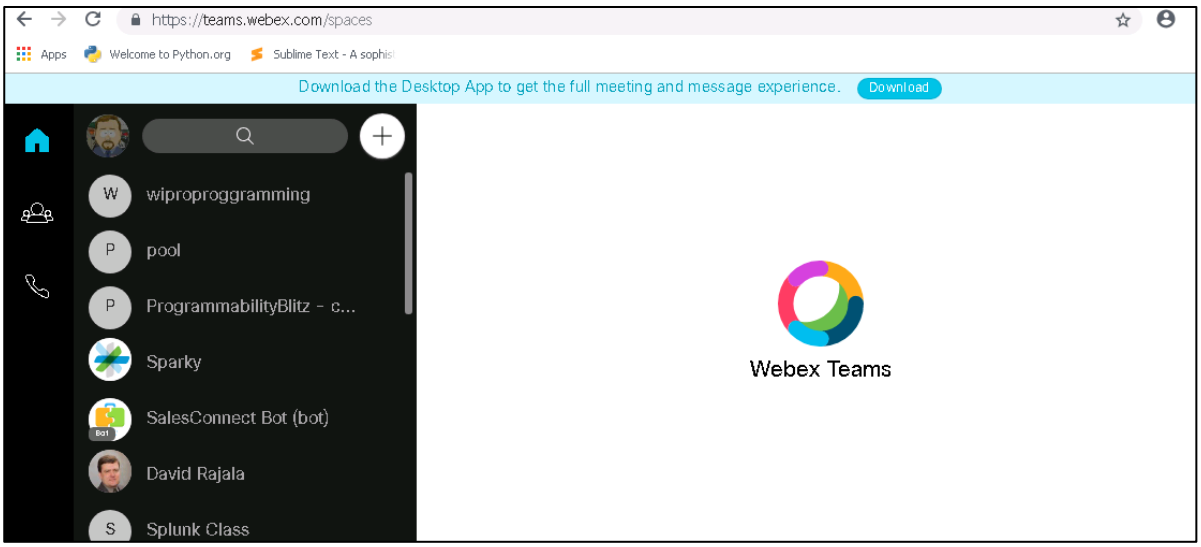
You will now log into your Webex Teams Account (create one if you have not yet created it using the “Join” option from the Menu).

**Step 1** Go to webex.com and click “Sign in” selecting “Teams”.




**Step 2** Log in with your credentials. Allow notification when asked. You should then see a page like below. Optionally, you can install the Desktop App.




# Task 4: Get your Webex Teams Access Token

## Activity

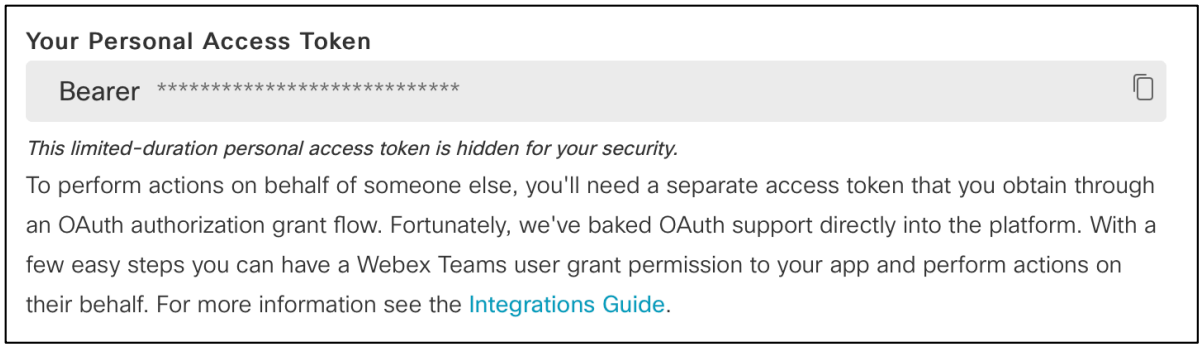
To interact with the Webex Teams APIs, you must have a **Webex Teams Access Token**. A Webex Teams Access Token is how the Webex Teams APIs validate access and identify the requesting user.

To get your personal access token, take the following steps

**Step 1** Login to developer.webex.com

**Step 2** Click on Docs or browse to the Getting Started page.

**Step 3** You will find your personal access token in the Accounts and Authentication section.




**Note** Your personal access token is great for testing the API with your account but it should never be used in production.

# Task 5: Use your Webex Teams Access Token

## Activity

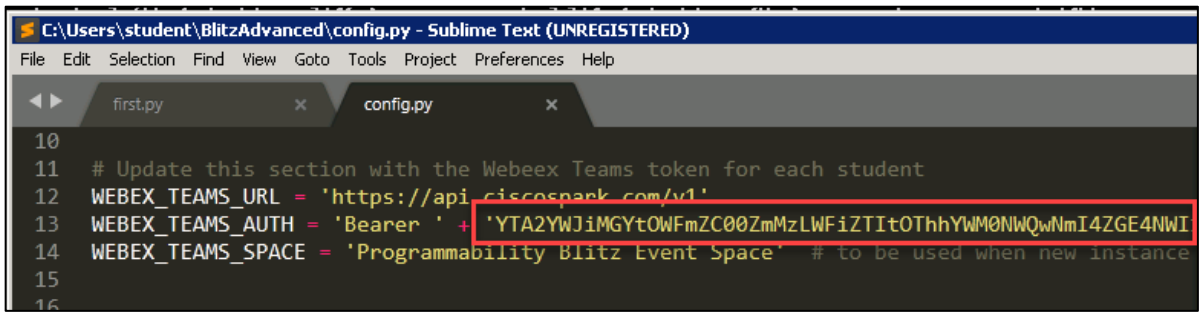
As a best practice, you can store your Webex Teams access token 'credential' as an environment variable in your development or production environment. By default, webexteamssdk will look for a WEBEX_TEAMS_ACCESS_TOKEN environment variable when creating new connection object.

There are many places and diverse ways that you can set an environment variable, which can include:

- A setting within your development IDE.
- A setting in your container / PaaS service.
- A statement in a shell script that configures and launches your app.

In our class we will be storing the token in a file called: **config.py**. This file is in the "*BlitzAdvanced-master*" folder.

**Step 1** Edit this file (you can use IDLE or your favorite text editor) and replace the WebEx Teams token with yours.



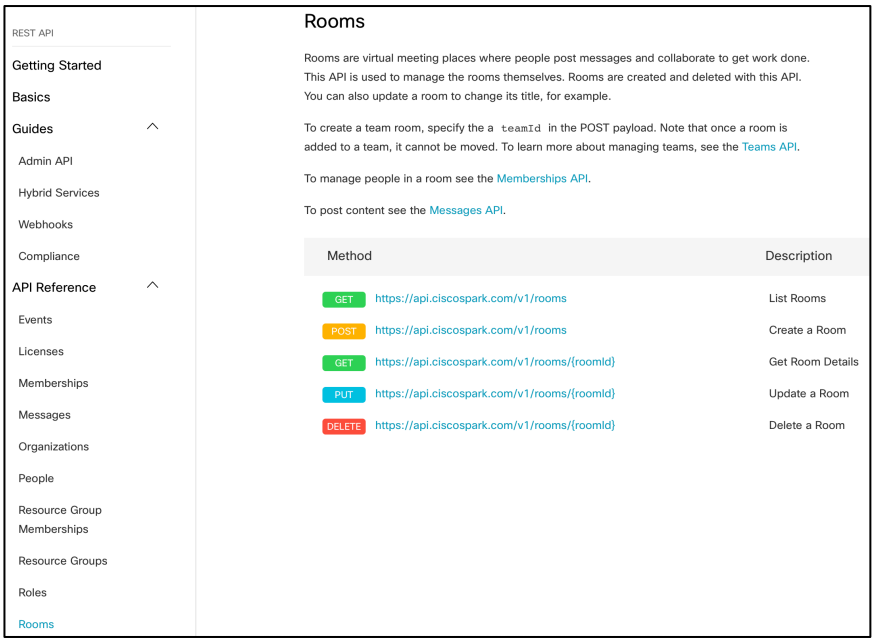

 

# Task 6: Listing Webex Teams Rooms

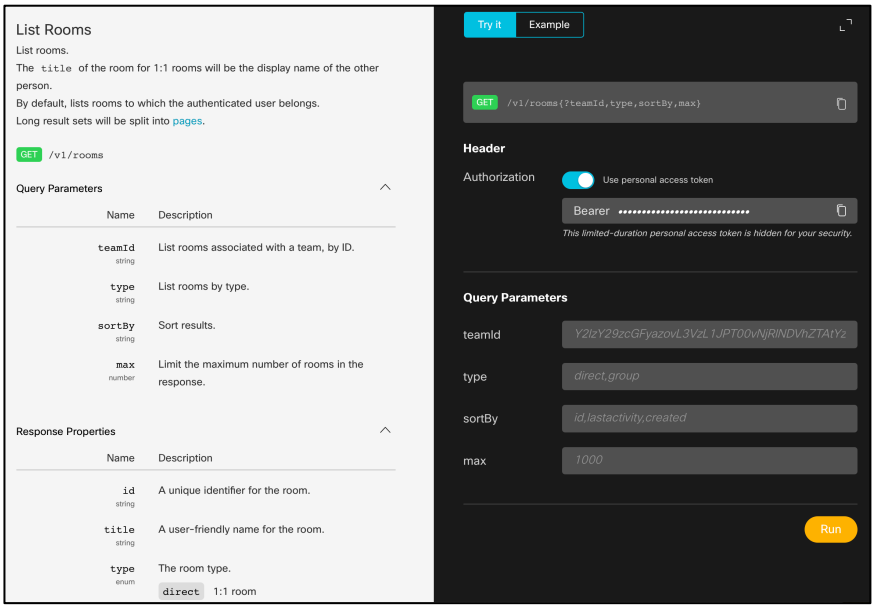
## Activity

The **GET REST** method is used to retrieve a listing or a single instance of a resource. Let's use the interactive docs to get a listing of your Webex Teams rooms.

**Step 1** To access the Rooms API docs, navigate to https://developer.webex.com and click on the **Documentation** option from the main menu, then expand the API Reference menu option and click on Rooms.




**Step 2** Click on the **GET** method.



**Step 3** Click on the **Run** button to execute the request, and verify the results (you may need to scroll down to view them).

# Advanced Path (Recommended) – Task 7: Writing the Python Code

**Step 1** Start a new script and Import the following packages:

- urllib3
- requests
- json
- sys
- webex_teams_apis

**Step 2** Import the following from the config.py file modified in Task 5

- WEBEX_TEAMS_URL
- WEBEX_TEAMS_AUTH

**Step 3** Define a function in the script that will create a Webex Teams room using the3 webex_teams_apis, the function to use is:

- **webex_teams_apis.create_space(space_name)**

**Step 4** Invoke the function only if the script was executed directly and not called within another script.

**Step 5** Test the script and make sure the new space was created within Webex Teams

# Foundation Path – Task 7: Writing the Python Code

## Activity

Now you will review and modify (if necessary) the Python Scripts we will use to interact with Webex Teams API.

Note that the interactive Doc Mode (from https://developer.webex.com) provides all of the REST API details we'll need to code a Python version of this operation, e.g.:

- **Method** - GET
- **URL** - https://api.ciscospark.com/v1/rooms
- **Headers**

- Content-type: application/json
- Authorization: Bearer ACCESS_TOKEN

- **Parameters**
  - teamId
  - max
  - type
  - sortBy

**Step 1** In the "*BlitzAdvanced-master*" folder, look for the webex_teams_module folder and open the **file webex_teams_start.py (use IDLE or any other text editor). Read through the code.**

```
1  #!/usr/bin/env python3
2
3  # developed by Gabi Zapodeanu, TSA, GPO, Cisco Systems
4
5  # import Python packages
6  import urllib3
7  import requests
8  import json
9  import sys
10 sys.path.append('..')
11
12 # import functions modules
13 import webex_teams_apis
14
15 from urllib3.exceptions import InsecureRequestWarning  # for insecure https warnings
16 from requests.auth import HTTPBasicAuth  # for Basic Auth
17 from config import WEBEX_TEAMS_URL, WEBEX_TEAMS_AUTH
18
19 urllib3.disable_warnings(InsecureRequestWarning)  # disable insecure https warnings
```

Line 6-9 import 4 Python Libraries.

Line 10 adds the parent directory to the search path (could also be done with virtual environments)

Line 13 imports the Webex Teams APIs package

Line 17 brings in definitions from the config.py file

Line 19, turns off some security warnings

```
20
21 def main():
22
23     # ask user to input a name for a new space
24     new_space_name = input('Enter a name for the Webex Teams Space: ')
25
26     # create a new space
27     space_id = webex_teams_apis.create_space(new_space_name)
28     print('The new space created has the Webex Teams id: ', space_id)
29
30
31 if __name__ == '__main__':
32     main()
33
```

Line 21 starts the main function

Line 24 asks the user to enter a name.

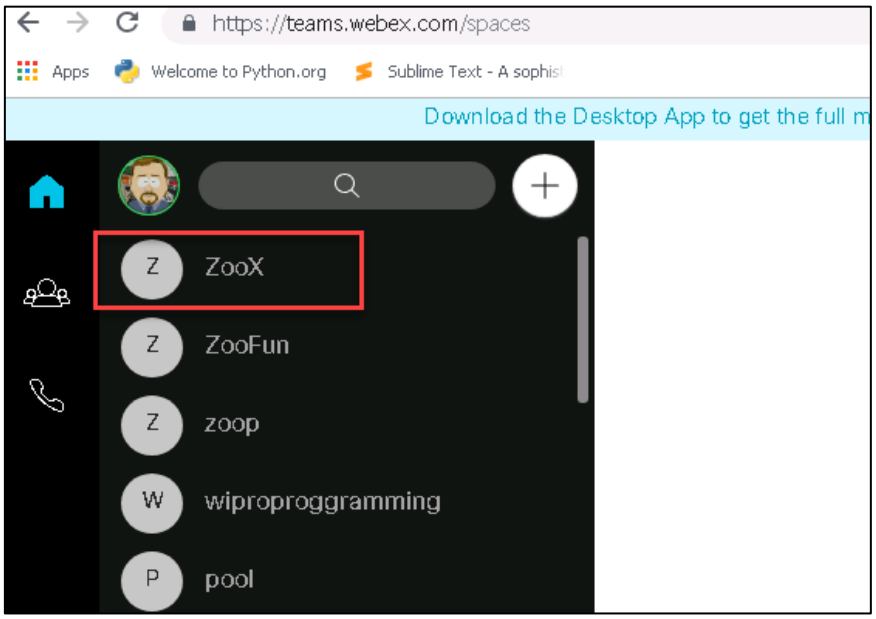
Line 27 creates the space.

Line 28 returns the ID of the space created

**Step 2** Now run the Python code and create a room (use any name you want. The example below is using ZooX where X is your POD number). From the command line:

```
C:\Users\student\BlitzAdvanced\webex_teams_module>python webex_teams_start.py
Enter a name for the Webex Teams Space: ZooX
The new space created has the Webex Teams id:  Y2lzY29zcGFyazovL3VzL1JPT00vYjE
TgwMzYtZDFlMDY2YjZiZDJk

C:\Users\student\BlitzAdvanced\webex_teams_module>
```

**Step 3** Go back to WebEX teams (web or application) and have a look at the newly created Room.




# Advanced Path (Recommended) – Task 8: POSTing a Message to Webex Teams

**Step 1** In your script that created a space, comment out the lines that create a new space.

**Step 2** Add to your script the functionality that prompts for the message to post to Webex Teams, then take that input and using the APIs post the message to the previously created space that was created in the previous task. Use the following function to post a new message

- **webex_teams_apis.post_space_message(new_space_name, new_message)**

# Task 8: POSTing a Message to WebEx Teams

## Activity

In this activity you will modify the Python Program to POST a message to the newly created Room.

**Step 1** Using the same **webex_teams_start.py** file, comment out the following lines by putting a '#' in front of them

```
# create a new space
# space_id = webex_teams_apis.create_space(new_space_name)
# print('The new space created has the Webex Teams id: ', space_id)
```

**Step 2** We will add a line to ask the user for what message to POST and a another line to POST the message:

```
new_message = input('\nEnter the message to be posted: ')
webex_teams_apis.post_space_message(new_space_name, new_message)
```

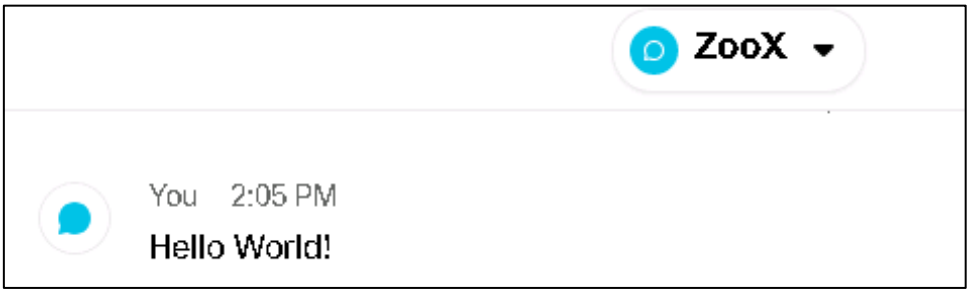
Your code should look similar to this:

```
def main():

    # ask user to input a name for a new space
    new_space_name = input('\nEnter a name for the Webex Teams Space: ')

    # create a new space
    # space_id = webex_teams_apis.create_space(new_space_name)
    # print('The new space created has the Webex Teams id: ', space_id)

    # ask user to post a message in a space
    new_message = input ('\nEnter the message to be posted in the space: ')
    webex_teams_apis.post_space_message(new_space_name, new_message)

    # ask user to input a name for a Domain and the correponding URL
    # new_domain_name = input('\nEnter the Text to be posted: ')
    # new_url = input('\nEnter the URL (format: "http://www..."): ')
    # webex_teams_apis.post_space_url_message(new_space_name, new_domain_name, new_url)

    # upload a new image file , one provided SDA_Roles.jpg
    # webex_teams_apis.post_space_file(new_space_name, 'SDA_Roles.jpg', 'image/jpg', '')
    # webex_teams_apis.delete_space(new_space_name)



if __name__ == '__main__':
    main()
```

**Step 3** Save and run the code. When prompted, enter the same Room Name you created before:

```
C:\Users\student\BlitzAdvanced\webex_teams_module>python webex_teams_start.py
Enter a name for the Webex Teams Space: ZooX

Enter the message to be posted in the space: Hello World!
```

**Step 4** Go back to WebEx teams and look for the message:




# Advanced Path (Recommended) – Task 9: POST a URL to the Room

**Step 1** In your script that you have been working with, comment out the lines that were used to post a new message

**Step 2** Add to your script the functionality that post a URL to the Webex Teams Space using the APIs. Prompt for the text of the message and the URL that will be used for that message. Use the following to post the link

- **webex_teams_apis.post_space_url_message(new_space_name, new_domain_name, new_url)**

# Foundational Path – Task 9: POST a URL to the Room

## Activity

In this task, you will modify the Python Program to POST a URL.

**Step 1** Comment out the lines from the previous task by putting a '#' in front of them.

**Step 2** Add the following lines:

```
new_domain_name = input('\nEnter the Text to be posted: ')
new_url = input('Enter the URL (format "http://..."): ')

webex_teams_apis.post_space_url_message(new_space_name, new_domain_name, new_url)
```

Your code should look similar to this:

```
def main():

    # ask user to input a name for a new space
    new_space_name = input('\nEnter a name for the Webex Teams Space: ')

    # create a new space
    # space_id = webex_teams_apis.create_space(new_space_name)
    # print('The new space created has the Webex Teams id: ', space_id)

    # ask user to post a message in a space
    # new_message = input ('\nEnter the message to be posted in the space: ')
    # webex_teams_apis.post_space_message(new_space_name, new_message)

    # ask user to input a name for a Domain and the correponding URL
    new_domain_name = input('\nEnter the Text to be posted: ')
    new_url = input('\nEnter the URL (format: "http://www..."): ')
    webex_teams_apis.post_space_url_message(new_space_name, new_domain_name, new_url)

    # upload a new image file , one provided SDA_Roles.jpg
    # webex_teams_apis.post_space_file(new_space_name, 'SDA_Roles.jpg', 'image/jpg', '')
    # webex_teams_apis.delete_space(new_space_name)



if __name__ == '__main__':
    main()
```

**Step 3** Save the file and run the code.

```
C:\Users\student\BlitzAdvanced\webex_teams_module>python webex_teams_start.py
Enter a name for the Webex Teams Space: ZooX

Enter the Domain Name be posted in the space: Funny
Enter the URL to be associated with the Domain Name (format "http://..."): http://www.yahoo.com
```

**Step 4** Go Back to Webex Teams and verify that the link was POSTed:


# Advanced Path (Recommended) – Task 10 and 11: POST an image to the Room

**Step 1** In your script that you have been working with, comment out the lines that were used to post a URL

**Step 2** Add to your script the functionality that post an image to the Webex Teams Space using the APIs. Use the following to post the image.

- **webex_teams_apis.post_space_file(new_space_name, 'SDA_Roles.jpg', 'image/jpg', '')**

**Step 3** After testing successfully delete the space previously created.

- **webex_teams_apis.post_space_file(new_space_name, 'SDA_Roles.jpg', 'image/jpg', '')**

# Task 10: POST an image to the Room

## Activity

In this activity we will modify the Python Program to POST an Image.

**Step 1** Comment out the lines from the previous task by putting a '#' in front of them.

**Step 2** Add the following lines:

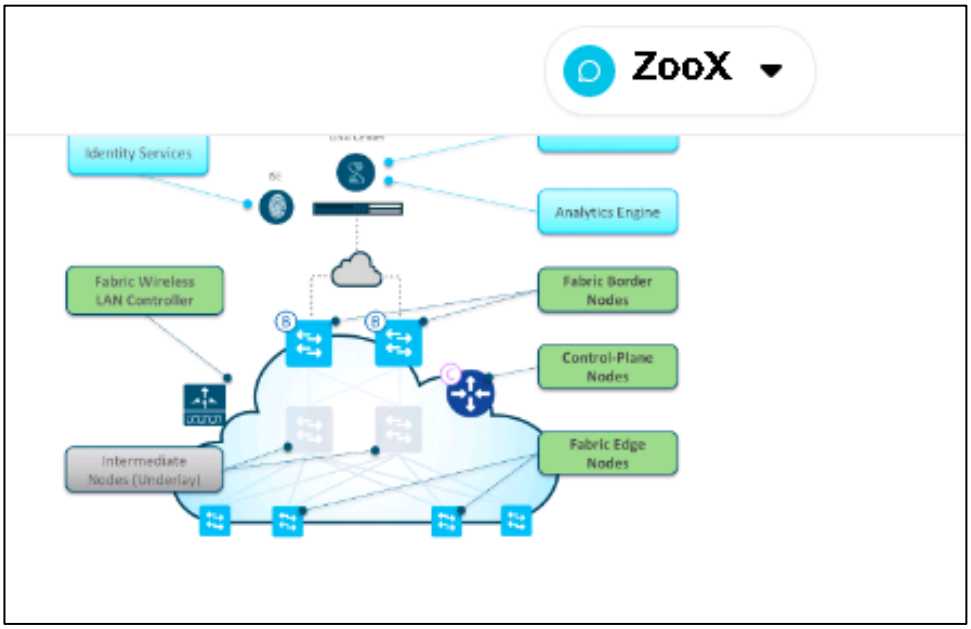
```
# upload a new image file, one provided SDA_Roles.jpg
webex_teams_apis.post_space_file(new_space_name, 'SDA_Roles.jpg', 'image/jpg',
'')
```

Your code should look similar to this:

```
def main():

    # ask user to input a name for a new space
    new_space_name = input('\nEnter a name for the Webex Teams Space: ')

    # create a new space
    # space_id = webex_teams_apis.create_space(new_space_name)
    # print('The new space created has the Webex Teams id: ', space_id)

    # ask user to post a message in a space
    # new_message = input ('\nEnter the message to be posted in the space: ')
    # webex_teams_apis.post_space_message(new_space_name, new_message)

    # ask user to input a name for a Domain and the correponding URL
    # new_domain_name = input('\nEnter the Text to be posted: ')
    # new_url = input('\nEnter the URL (format: "http://www..."): ')
    # webex_teams_apis.post_space_url_message(new_space_name, new_domain_name, new_url)

    # upload a new image file , one provided SDA_Roles.jpg
    webex_teams_apis.post_space_file(new_space_name, 'SDA_Roles.jpg', 'image/jpg', '')
    webex_teams_apis.delete_space(new_space_name)



if __name__ == '__main__':
    main()
```

**Step 3** Execute the code and check Webex Teams for the image




# Task 11: Delete the Room

## Activity

In this activity we will modify the Python Program to delete the Room you created.

**Step 1** Comment out the lines from the previous task by putting a '#' in front of them.

**Step 2** Add the following lines:

```
# delete the created room

webex_teams_api.delete_space(new_space_name)

print ("Room '" + new_space_name + "' has been deleted.")
```

# Common Path for Advanced and Foundation – Task 12: Explore and Run another Python Program

## Activity

In this activity you will review and run a Python Script available in the webex_teams_module folder.

**Step 1** Open webex_teams_lab1.py using Atom.

**Step 2** Take some time and review the code.

```
# create a new space
space_id = webex_teams_apis.create_space(new_space_name)
print('The new space created has the Webex Teams id: ', space_id)

# ask user to input an email for a Webex Teams user
new_membership = input('\nEnter an email for a Webex Teams user to invite to the space: ')

# invite user to join the space
webex_teams_apis.add_space_membership(new_space_name, new_membership)
input('Check the space for the new membership, hit any key to continue ')

# ask user to input the message to be posted in the space
new_message = input('\nEnter the message to be posted in the space: ')

# post message to the new space
webex_teams_apis.post_space_message(new_space_name, new_message)
input('Check the space for the new message posted, hit any key to continue ')

# post the same message marked down
webex_teams_apis.post_space_markdown_message(new_space_name, new_message)
input('\nCheck the space for the marked down message posted, hit any key to continue ')

# ask user to input an domain name, and the URL link to be posted in the space
new_domain_name = input('\nEnter the Domain Name be posted in the space: ')
new_url = input('Enter the URL to be associated with the Domain Name (format "http://...")

# post the new Domain Name and URL to the space
webex_teams_apis.post_space_url_message(new_space_name, new_domain_name, new_url)
input('Check the space for the URL posted, hit any key to continue ')

# upload a new image file, one provided SDA_Roles.jpg
webex_teams_apis.post_space_file(new_space_name, 'SDA_Roles.jpg', 'image/jpg', '')
input('\nCheck the space for the image file uploaded, hit any key to continue ')

# ask the user invited to post a message to the space
input('\nAsk the user invited to the space to post a message in the space using the Mobile

# check for the last message posted in the space and the identity of the user
last_message_info = webex_teams_apis.last_user_message(new_space_name)

print('\nLast message posted in the space was - ', last_message_info[0], '\nLast message w

# delete the space
input('\n\nEnter any key to delete the space you created ')

webex_teams_apis.delete_space(new_space_name)
```

**Step 3** Execute the program and review the output in WebEx Teams.

# Lab 4a: Meraki Automation with Python


## Introduction

You will be creating/using Python Programs to achieve the intended results.

## Advanced Path (Recommended) – Task 1: Install Needed Modules

### Activity

In order to be able to use the Cisco Meraki API capabilities in a Python Script, we need to install some modules that will provide access to the functions we can use within Cisco Meraki Dashboard. The modules we need are **requests** and **meraki**. Make sure you have them installed on your Student Desktop.

**Step 1** Login the Student Desktop using http://labs.nocular.com with credentials provided by the instructor and then go to the Ubuntu Student Desktop**.**

**Step 2** Install the requested modules and verify the installation using the Python Interactive Prompt and the following Meraki Key "e24759c28edd1d97715a6ba9ea8bc679c5d2706b" with the **myorgaccess** function, contained in the **meraki** module, to obtain the Org Id.

**Step 3** Output should be similar to the one below:

```
[{'id': '578149602163687854', 'name': 'Public API Lab'}]
```

# Task 2: Python Script

**Step 1** Create a Python Script to output the list of networks and the devices contains on them for the organization retrieved on Task 1 and store the output in a CSV file.

# Task 3: Dashboard API Python

**Step 1** Using a browser, access the Meraki Dashboard API Python Module on the following URI:

- **https://github.com/meraki/dashboard-api-python/blob/master/meraki.py**

**Step 2** Using the Meraki Dashboard API Python Module, build a Python Program that:

- Creates your network in the Organization
- Queries the Org's overall inventory
- Adds a device from the inventory to your network
- Updates the location of that device, and finally
- Updates the network's SSID config (15 SSDIs in total).

**Optional Bonus Mission** (Difficulty Level - ???)

-

**Optional Bonus Mission** (Difficulty Level – ???

-

# Foundation Path – Task 1: Install Needed Modules

## Activity

In order to be able to use the Cisco Meraki API capabilities in a Python Script, we need to install some modules that will provide access to the functions we can use within Cisco Meraki Dashboard. The modules we need are **requests** and **meraki**.

**Step 1** Open a terminal window and install the requested modules using the following commands:

- pip install --upgrade requests
- pip install --upgrade meraki

**Step 2** To make sure your installation is successful, launch Python Interactive Prompt using the "python" command from a CMD Window or via the Python Application Menu.

**Step 3** To make sure your installation is successful, launch a Python Interactive

**Step 4** Now you should see the ">>>" prompt.

**Step 5** From the Python Interactive Prompt, enter the following lines of code:

```
from meraki import meraki
meraki.myorgaccess("e24759c28edd1d97715a6ba9ea8bc679c5d2706b")
```

You should see similar results being returned (Organization ID and Name):

```
[{'id': '578149602163687854', 'name': 'Public API Lab'}]
```

# Task 2: Python Script

## Activity

This is a simple exercise in which you run a pre-written Python Program that gathers the networks and devices in the "Public API Lab" Organization. The script outputs the information to two CSV files, one for appliances, and another for all other devices.

**Step 1** Open the **uplink.py** Python file (e.g., using the "IDLE" Python Editor), located in the Meraki folder within the **BlitzAdvanced-master** folder.

**Step 2** Scan through the file, looking at the comments (lines beginning with # sign) heading each section, to get a rough sense of the script's workflow. Please change the **API_KEY** to the string showed on Task 1 – Step 4.

**Step 3** Run from the CMD the Python Program using the following command:

- python uplink.py

Output should look similar to this:

```
C:\Users\student\BlitzAdvanced\Meraki>python uplink.py
Looking into network Kiwi
Found device Kiwi
Looking into network Bas Bannink
Found device Bas Bannink
Looking into network nick
Found device nick
Looking into network joe
Found device Q2ED-MLNT-WLBV
```

# Task 3: Dashboard API Python

## Activity

This exercise is intended to get your feet wet with using the Meraki Dashboard API Python module with a mostly pre-written script that you will need to edit, with some blanks to fill!

The exercise walks you through building a script that 1) Creates your network in the Organization, 2) Queries the org's overall inventory, 3) Adds a device from the inventory to your network, 4) Updates the location of that device, and finally, 5) Updates the network's SSID config.

*You will only need to edit the sections (12 lines total) of the script between the "START EDITING" and "END EDITING" comment blocks. Do NOT change anything outside of those sections.*

To get started, take the following steps:

**Step 1** Open lab_public.py in IDLE or any Text Editor.

**Step 2** Open a CMD Window and go to the Meraki Folder.

**Step 3** In a browser, open the Dashboard API module meraki.py file on GitHub

- **https://github.com/meraki/dashboard-api-python/blob/master/meraki.py**

You will go back and forth between editing the lab_public.py script and running it, looking up the module's source code for specific functions. All configuration should be done through API Calls.

# Task 4: Part 1. Create a Network

## Activity

In this activity you will use the Cisco Meraki Dashboard API capabilities to create an object called network.

**Step 1** Let's start off easy! For this first part, all you need to do is fill in the variable section with your name, some tags to apply to the network, and the Preferred Time Zone (lab_public_py).

**Step 2** Note that this part calls two functions: *getnetworklist* with the API key and ORG ID, along with *addnetwork* in the module (search for "# Create a network"). The latter function passes in the same two parameters as the former, along with your name as the network name, str (that's the Python string type) 'wireless' for an MR network, the Tags, and Time Zone.

```
#########################
##### START EDITING #####
my_name = 'First Last'
my_tags = ['Tag1', 'Tag2', 'Tag3']
my_time = 'US/Pacific'
###### END EDITING ######
#########################
```

# Task 4: Part 2. Return the Inventory

## Activity

In this activity you will focused on calling the Inventory within the Cisco Meraki Dashboard.

**Step 1** Here, you'll need to figure out how to call the getorginventory function. Refer to the function definition in the module, or you can also run "help(meraki.getorginventory)" in the Python interactive interpreter (after importing first with "from meraki import meraki").
In terminal/command prompt, run "python"; this starts the Python Interpreter, then:

**Step 2** Using the Python Interactive Prompt (simply launch python from a CMD Window typing "python"), type the following lines of code (remind that you need to open the CMD Window and "change directory" to be inside the Meraki folder:

```
from meraki import Meraki
help(meraki.getorginventory)
```

**Step 3** The list comprehension then assigns all devices. Add two parameters: API key and orgID.

```
#########################
##### START EDITING #####
# Call to return the inventory for an organization
# One line similar to above line 33's call on meraki.getnetworklist()
inventory = meraki.getorginventory(my_key, my_org)
###### END EDITING ######
#########################
```

# Task 4: Part 3. Claim Device into Network

## Activity

In this activity you will claim a device not part of any network into the network you created before in Task 4 Part 1.

**Step 1** This part of the script randomly selects one of the unused APs from inventory, and now you will need to call another function to claim the device into your wireless network.

**Step 2** You will only claim that one device into the network, no more.

```
#########################
##### START EDITING #####
# Call to claim a device into a network
# Only one line needs to be filled here, with the same indentation (4 spaces)
meraki.adddevtonet (my_key, my_org, my_serial)
###### END EDITING ######
#########################
```

# Task 4: Part 4. Update Device Properties

## Activity

In this activity you will update the location of the device you previously imported into you network.

**Step 1** What's your favorite/next vacation spot? Enter it in for the str variable my_address!

**Step 2** Now update your device's name to be the same as the network name, tags same as network tags, and location to your vacation address, while moving the map maker.

**Tip** You do not have to worry about entering latitude/longitude coordinates. Double-check the module's function (search for "# Update the attributes of a device") for the exact parameters needed, along with the parameters as specified by the Dashboard API Automation with Python – API Labdocs.

# Task 4: Part 5. Update first SSID config

## Activity

You will now update the first SSID with all the necessary parameters, based on Cisco Meraki API requirements.

**Step 1** To configure a WPA2-Personal SSID into the first SSID slot, input a different name (than earlier) for your SSID and a PSK with at least eight characters.

**Step 2** Make sure that the SSID is enabled. (Hint: the number for the first SSID is 0).

# Task 4: Bonus Part: Update Multiple SSIDs' Config

## Activity

Care to repeat the previous part but now for the rest of the 14 SSIDs? Give it a try if you want.

**Step 1** With two lines, change the configuration for the other 14 SSIDs as well.

**Step 2** The built-in range function can help here; try calling "help(range)" in the Python Interactive Prompt to see its usage.

**Step 3** The purpose of this last part is to practice using a for loop within Python, along with string concatenation and type conversion (changing a data variable from one type to another). Why would these methods resolve the "Bad Request" return with a status of 400?

### Success!

Congratulations! You're now on your way to Automating the Meraki Dashboard API with Python!

Refer to the lab_answers.py file for a working solution and check your work.

# Lab 4b: Meraki Dashboard Reports with Google Sheets


## Introduction

This Lab will provide you visibility into integrating Cisco Meraki Dashboard API with Google Sheet. It is intended to serve as an example in terms of the possibilities APIs provide. Easily import Meraki Network data into Google Sheets using the Meraki Dashboard API.

# Advanced Path (Recommended) – Task 1: Dashboard API Script Reports Demo

## Activity

In this activity you will use the embedded capabilities of the Google Sheet.

**Step 1** Open the Google Sheet available in the following links:

- Meraki-Reports Sheet or use the following Tiny URL http://nr4.us/blitz23

**Step 2** Then copy the file to your own account.

# Task 2: Get the Organization ID and API Key

## Activity

In this task you will get the Organization ID using the appropriate API Key in order to authorize appropriately the Google Sheet Code execution within Cisco Meraki Dashboard API.

**Step 1** Use the following API Key on the Settings Tab:

"e24759c28edd1d97715a6ba9ea8bc679c5d2706b"

# Task 3: Run Reports

**Step 1** Feel free to run some reports and have a look at the results.

# Foundation Path – Task 1: Dashboard API Script Reports Demo

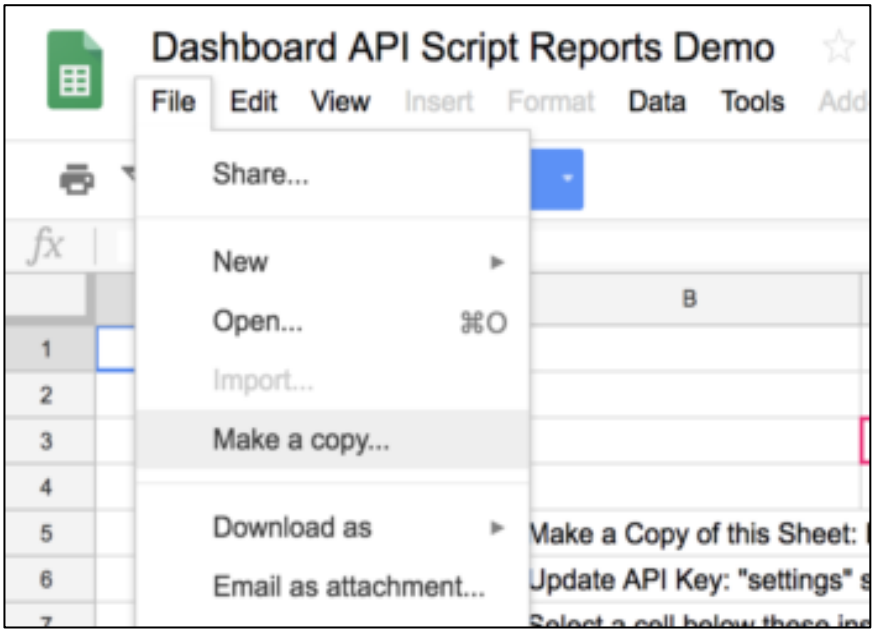
## Activity

In this activity you will use the embedded capabilities of the Google Sheet.

**Step 1** Simply run this demo by opening the Google Sheet available following the embedded link here:

Meraki-Reports Sheet or use the following Tiny URL http://nr4.us/blitz23

**Step 2** Then copy the file to your own account by selecting File > Make a Copy. This will allow the script to run and ensure your credentials are kept within your account.

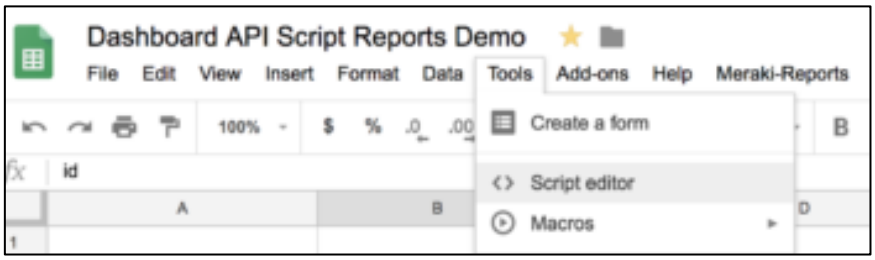


# How does it work? (just read below)

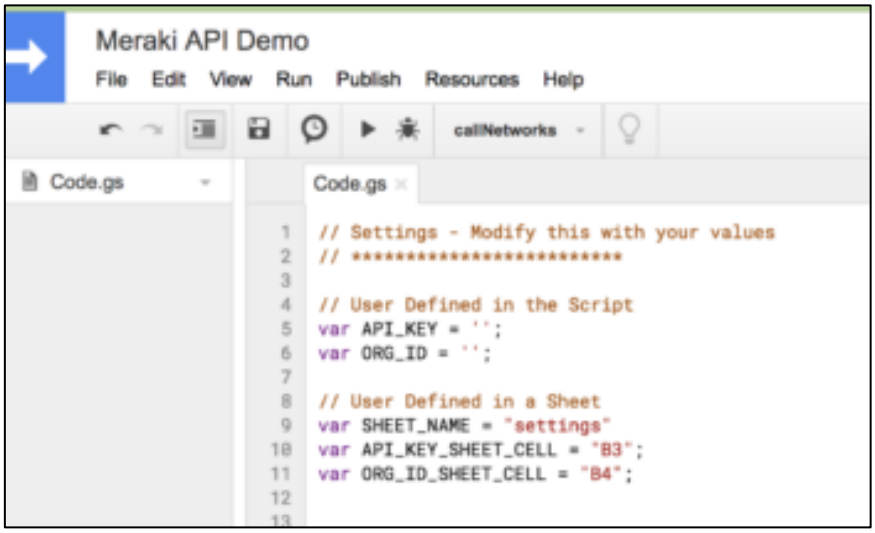
The Google Sheet has a Code.gs file attached to it. This script creates a menu of reports that run functions to call the Meraki Dashboard API.

Follow this link to github.com to see the content of the file (Code.gs)

You can find this code in Tools > Script editor




The beginning of the script includes variables that can be adjusted by the user. Enter your Meraki Dashboard API key here as well as the default organization ID. You can also adjust which sheet and cell these variables can be pulled from.




# Script Details

If you would like to extend the reports you can modify the JavaScript file as needed. Here is a general overview of what each code section does.

# Dashboard API Functions

These are the individual calls to the Meraki Dashboard API. To add additional Meraki endpoints copying a similar function and adjust the path and function name.

```
function getOrgs(apiKey) {
  var response = UrlFetchApp.fetch("https://api.meraki.com/api/v0/organizations",
{headers:{'X-Cisco-Meraki-API-Key': apiKey}});
  var data = response.getContentText();
  var json = JSON.parse(data);
```

```
  return json;
```

## Report Functions

These functions will make the request to the API functions and format the data as necessary. You will define what parameters of the object should be set as the column headers. These names should match the JSON object key for the respective value.

```
function callOrgs(){
  var data = getOrgs(settings.apiKey);
  displayJSON(data, ['id','name']);
}
```

If there are nested JSON objects, some additional logic will need to be used to flatten the data.

```
function callLicenseState(){
  var data = getLicenseState(settings.apiKey, settings.orgId);

  // flatten model names and values
  var models = Object.keys(data.licensedDeviceCounts);
  models.forEach(
    function (m){
      data[m] = data.licensedDeviceCounts[m];
    }
  );
  var keys = ['status', 'expirationDate'];
  var combinedKeys = keys.concat(models);
  displayJSON([data], combinedKeys);
}
```

## Toolbar Menu

This is where we define what menu options are available to the Sheets page and which function should be called for each.

```
// Toolbar Menu Items
function onOpen() {
  var ui = SpreadsheetApp.getUi();
  ui.createMenu('Meraki-Reports')
      .addItem('Organizations for API key','callOrgs')
      .addItem('Organization','callOrg')
      .addItem('Inventory','callInventory')
```

```
      .addItem('Networks','callNetworks')
      .addItem('Devices','callDevices')
      .addItem('License State','callLicenseState')
      .addItem('Configuration Templates','callConfigTemplates')
      .addItem('Group Policies','callGroupPoliciesOfOrg')
      .addItem('SSIDs','callSsidsOfOrg')
      .addItem('VLANS','callVlansOfOrg')
      .addToUi();
```

## Display Functions

This function is responsible for converting the data arrays into the Google sheet format. It handles the ability to place the report data where the active cell is. You shouldn't have to modify this.

```
function displayJSON(json, keys){
  //json = [{"id":"1234","name":"sample"},{"id":"9876","name":"sample 2",
"extra":"more info"}];
  Logger.log('displayJSON'+ JSON.stringify(json));
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheets = ss.getSheets();
  var sheet = ss.getActiveSheet();
  ...
```

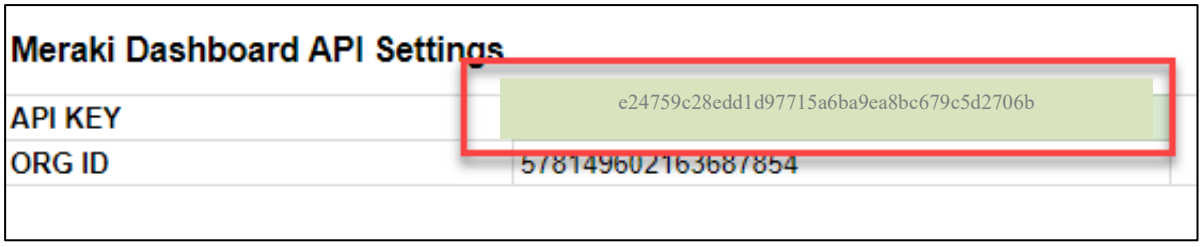
# Task 2: Get the Organization ID and API Key

## Activity

In this task you will get the Organization ID and API Key in order to authorize appropriately the Google Sheet Code execution within Cisco Meraki Dashboard API.

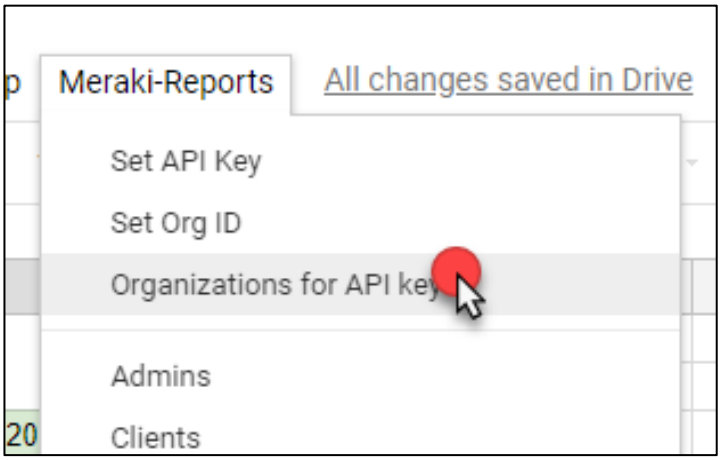
**Step 1** Use the following API Key on the Settings Tab:

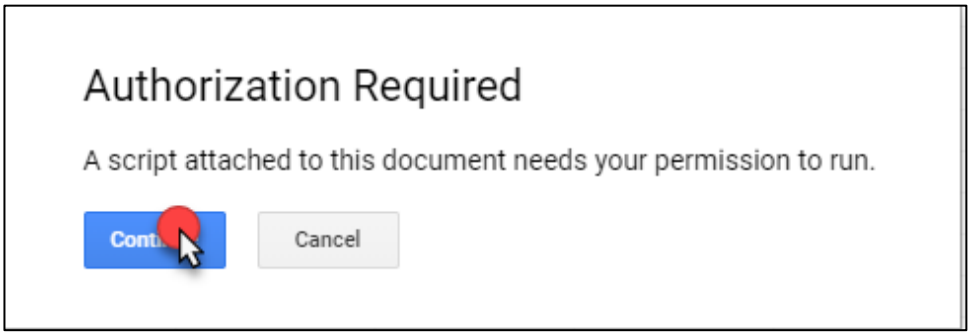
"e24759c28edd1d97715a6ba9ea8bc679c5d2706b"

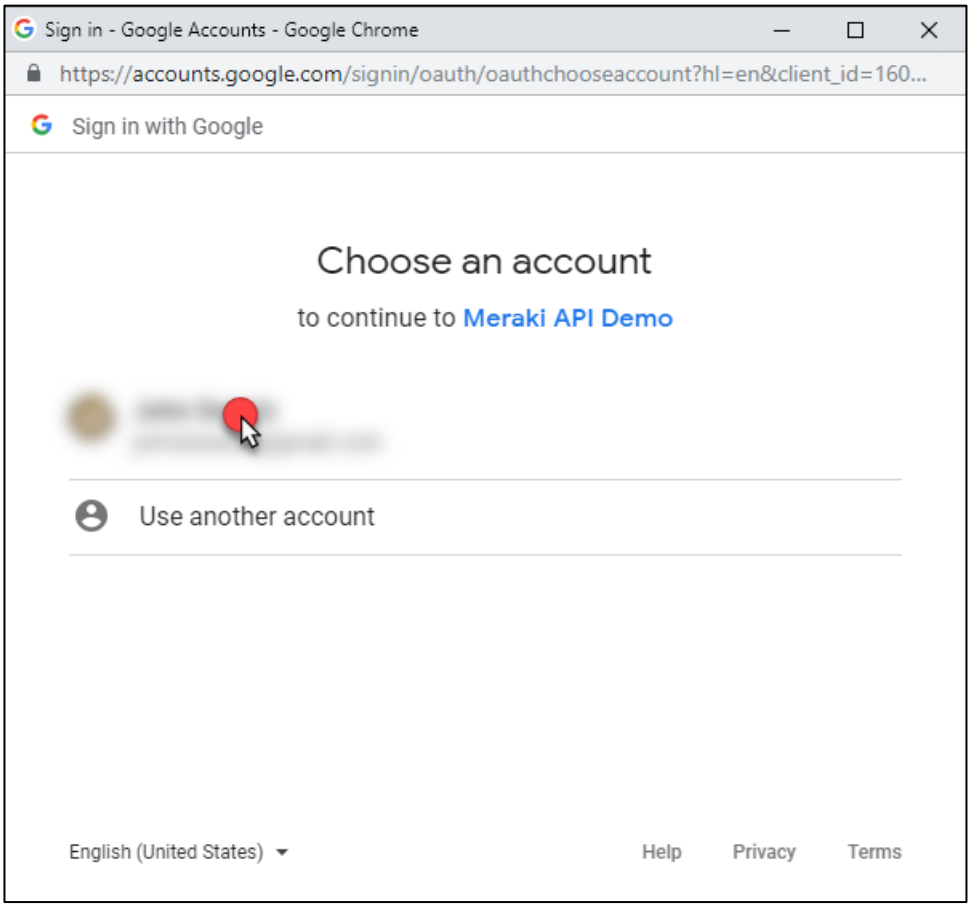


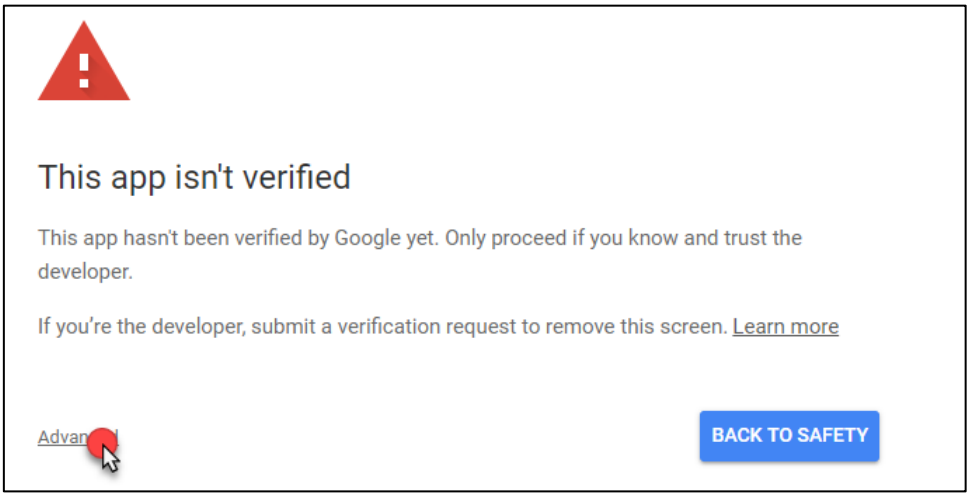

**Step 2** Get the Organizations for the API key you just configured.

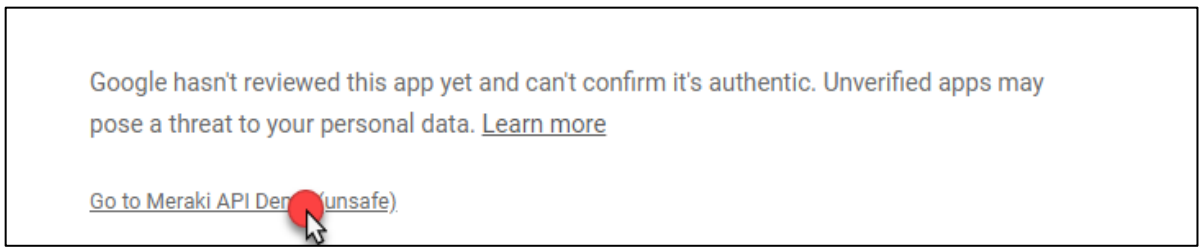
 

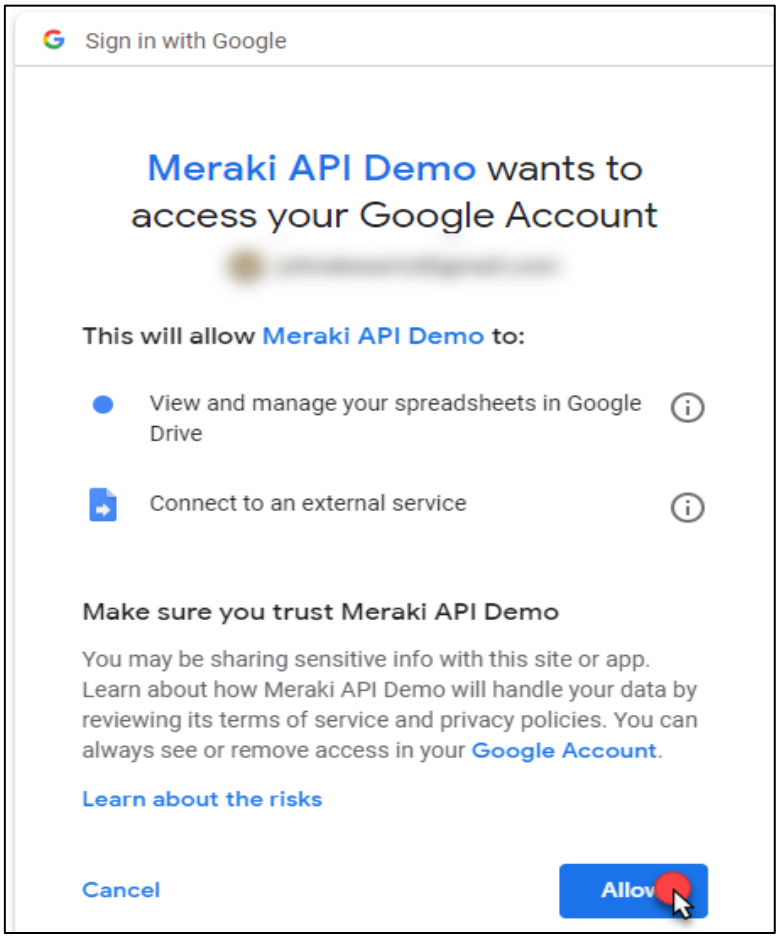

p
Meraki-Reports
All changes saved in Drive
Set API Key
Set Org ID
Organizations for API key
Admins
20
Clients


Authorization Required
A script attached to this document needs your permission to run.
Cancel


Sign in - Google Accounts - Google Chrome
https://accounts.google.com/signin/oauth/oauthchooseaccount?hl=en&client_id=160...
Sign in with Google
Choose an account
to continue to Meraki API Demo
Use another account
English (United States)
Help
Privacy
Terms



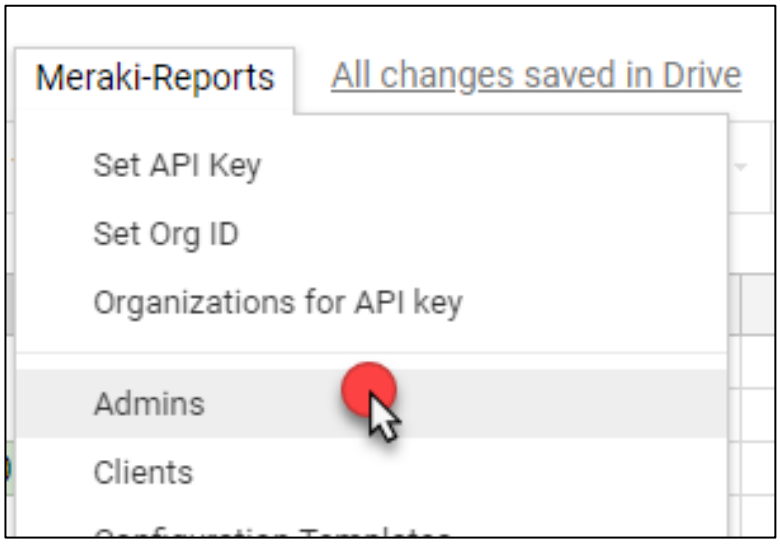
| id | name |
| --- | --- |
| 578149602163687854 | Public API Lab |

# Task 3: Run Reports

## Activity

Now that you have configured proper authorization to Google Sheet with Cisco Meraki Dashboard API, you can start takin advantage of the Google Sheet capabilities.

**Step 1** How many Admins?




**Step 2** Run some other reports (Warning, some of them can take a while)

# Lab 5: IOS XE Programming

## Introduction

IOS XE is a powerful OS that many Cisco Catalyst and Router Devices use. It has been enhanced with functionality to allow integration into the Software-Defined World via open protocols and Data Models like NETCONF, RESTCONF, YANG. It also supports containers, due to it Unix-Based approach, which means you can install native Linux APPs within the networking device. You can make use of the embedded Python Module and run scripts on-box. In this lab you will deploy a solution that allows for change control of router configurations by invoking EEM to run a Python script when a change is detected. This script will open a ticket in ServiceNow and then create a Webex Teams space, add a user to the space, send a message asking for approval for the change and if approved the new configuration will become the baseline configuration. If not approve any changes will be rolled back. The Webex Teams space will then be deleted when done.

## Common Task 1: Setting up the CSR1000V for Guestshell (completed by all paths)

Connect to your assigned CSR1000V and configure AAA, a virtual port group for the Guestshell connectivity, NAT with the Guestshell, then enable guestshell and test connectivity.

## Activity

**Step 1** SSH to your assigned CSR1000v using the username **admin** and the password **ISEisC00L**

**Step 2** Configure IOX on from the configuration mode of the CSR1000V by entering the command **iox**.

**Step 3** Confirm that IOX is running by issuing the **show iox** command from privileged exec mode. This may take a minute for the services to show running.

```
ABlitz#show iox
Virtual Service Global State and Virtualization Limits:

Infrastructure version : 1.7
Total virtual services installed : 1
Total virtual services activated : 1

Machine types supported   : LXC
Machine types disabled    : KVM

Maximum VCPUs per virtual service : 0
Resource virtualization limits:
Name                         Quota     Committed     Available
------------------------------------------------------------------
system CPU (%)                  75             5            70
memory (MB)                   1024           512           512
bootflash (MB)               20000           756          4517


IOx Infrastructure Summary:
---------------------------
IOx service (CAF)    : Running
IOx service (HA)     : Not Running
IOx service (IOxman) : Running
Libvirtd             : Running
```

**Step 4** Configure the following AAA commands from configuration mode

- **aaa new-model**
- **aaa authentication login default local**
- **aaa authorization exec default local**

```
ABlitz(config)#aaa new-model
ABlitz(config)#aaa authentication login default local
ABlitz(config)#aaa new-model
```

**Step 5** Set a default route with **ip route 0.0.0.0 0.0.0.0 172.16.3.1**

**Step 6** Set a DNS server with the command **ip name-server 8.8.8.8**

```
ABlitz(config)#ip route 0.0.0.0 0.0.0.0 172.16.3.1
ABlitz(config)#ip name-server 8.8.8.8
```

**Step 7** a Virtual port group 0 with the command **interface VirtualPortGroup0**

**Step 8** Assign the following to the new Virtual Port Group

- **ip address 10.1.1.1 255.255.255.0**
- **ip nat inside**
- **description Used_for_GS_access**

```
ABlitz(config-if)#interface VirtualPortGroup 0
ABlitz(config-if)#ip add 10.1.1.1 255.255.255.0
ABlitz(config-if)#ip nat inside
ABlitz(config-if)#description Used_for_GS_Access
```

**Step 9** Set the outside NAT interface on Interface Gigabit Ethernet 2 with the command **ip nat outside** (the ip address will be unique to the pod)

```
interface GigabitEthernet2
 ip address 172.16.3.111 255.255.255.0
 ip nat outside
```

**Step 10** Configure the ACL that will be used for NAT with the commands

- **ip access-list extended Internet_ACL**
- **deny ip host 10.1.1.2 10.10.10.0 0.0.0.255**
- **permit ip host 10.1.1.2 any**

```
ip access-list extended Internet_ACL
 deny   ip host 10.1.1.2 10.10.10.0 0.0.0.255
 permit ip host 10.1.1.2 any
```

**Step 11** Next enter **ip nat inside source list Interne_ACL interface GigabitEthernet 2**

**Step 12** Configure the ACL that will be used for NAT with the commands

- **ip access-list extended Lab_ACL**
- **permit ip host 10.1.1.2 10.10.10.0 m0.0.0.255**

```
ip access-list extended Lab_ACL
 permit ip host 10.1.1.2 10.10.10.0 0.0.0.255
```

**Step 13** Next enter the command **ip nat inside source list Lab_ACL interface GigabitEthernet 1**

**Step 14** Enter **app-hosting appid guestshell**

**Step 15** In the app sub configuration mode enter **vnic gateway1 virtualportgroup 0 guest-interface 0 guest-ipaddress 10.1.1.2 netmask 255.255.255.0 gateway 10.1.1.1 name-server 8.8.8.8**

**Step 16** Finally from privileged exec mode enter **guestshell enable** to start guestshell, be patient this may take a bit

**Step 17** Verify that the guestshell is started properly by issuing the **show app-hosting list** command

```
ABlitz#sho app-hosting detail
State                  : RUNNING
Author                 : Cisco Systems
Application
  Type                 : lxc
  App id               : guestshell
  Name                 : GuestShell
  Version              : 2.1(0.1)
Activated profile name : custom
  Description          : Cisco Systems Guest Shell XE for x86
Resource reservation
  Memory               : 512 MB
  Disk                 : 1 MB
  CPU                  : 799 units
  VCPU                 : 1
```

**Step 18** Verify connectivity is configured correctly by running **guestshell run sudo ping www.cisco.com**

**Step 19** Verfify connectivity with our DNA Center by using **guestshell run sudo ping 10.10.10.204**

# Task 2: Guest Shell Configuration

Configure the GuestShell with the required packages and applications

## Activity

In this activity you will complete the necessary steps to have Guest Shell enable on the networking device.

**Step 1** From your CSR1000V start as bash shell in guestshell by running the command **guestshell run bash**

**Step 2** From the bash shell load the following packages:

- **sudo pip install –upgrade pip**
- **sudo pip install requests**
- **sudo pip install requests[security]**
- **sudo pip ncclient==0.6.3**

**Step 3** Install and configure git (you will need a Github username and password) with the following commands:

- **sudo yum install git -y**
- **git config --global user.name "your Github username"**
- **git config --global user.email "youremail@yourdomain.com"**
- **git config  --global credential.helper "cache -timeout 3600"**
- **git config --list**

**Step 4** Install some additional packages with the following commands:

- **sudo yum install curl-devel -y**
- **sudo yum update -y nss curl libcurl**
- **sudo yum install nano -y**

**Step 5** Change to the bootflash directory with by entering **cd /bootflash**

**Step 6** Load the class files for this demo from github by using the command **git clone 'https://github.com/deplorablejed/Adv-Prog-Blitz/ChatOps**

**Step 7** Verify the files replicated correctly by entering the command **ls /bootflash/ChatOps**

# Common Task 3: Set ChatOps Variables

## Activity

Configure the required variables in the config.py script using the Guestshell nano editor

**Step 1** From your Student Desktop get your personal Webex Teams token by going to http://developer.webex.com and then navigating to Documentation → Getting Started → Copy your token to your clipboard.

**Step 2** From your CSR1000V start as bash shell in guestshell by running the command **guestshell run bash**

**Step 3** Change to the /bootflash/ChatOps directory and use the command **nano config.py**. You should now be able to edit the variables in this file. Confirm the settings and if necessary set them as follows:

- **WEBEX_TEAMS_URL = 'https://api.ciscospark.com/v1'**
- **WEBEX_TEAMS_AUTH = 'Bearer ' + YOUR**
- **WEBEX_TEAMS_SPACE = 'ChatOps'**
- **WEBEX_TEAMS_MEMBER = 'YOUR_EMAIL_HERE'**
- **SNOW_URL = 'https://devXXXXX.service-now.com/api/now'** (get XXXXX from your instructor)
- **SNOW_USER = 'IOSXE'**
- **SNOW_PASS = 'IOSXE'**
- **SNOW_INSTANCE = 'devXXXX'** (get XXXXX from your instructor)
- **HOST = '10.10.10.20*X*'** (where ***X*** is your pod number)
- **USER = 'admin'**
- **PASS = 'ISEisC00L'**
- **PORT = '830'**

```
WEBEX_TEAMS_URL = 'https://api.ciscospark.com/v1'
WEBEX_TEAMS_AUTH = 'Bearer ' + 'Mjg3NmI5OGQtNjZkOS00YTkyLTkz
WEBEX_TEAMS_SPACE = 'TestSpace'
WEBEX_TEAMS_MEMBER = 'jedemeul@cisco.com'


SNOW_URL = 'https://dev64635.service-now.com/api/now'
SNOW_USER = 'IOSXE'
SNOW_PASS = 'IOSXE'
SNOW_INSTANCE = 'dev64635'


HOST = '10.10.10.209'
USER = 'admin'
PASS = 'ISEisC00L'
PORT = '830'
```

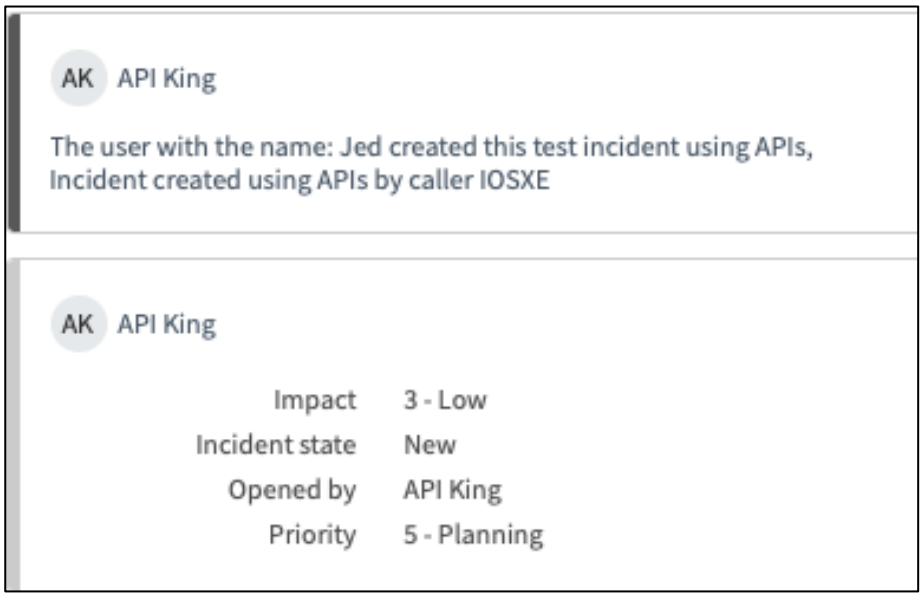
# Common Task 4: Run Apps on the Guestshell

**Step 1** From your CSR1000V start as bash shell in guestshell by running the command **guestshell run bash**

**Step 2** Change to the /bootflash directory and use the command **mkdir CONFIG_FILES** to add a folder. confirm the folder exists by using the ls command

**Step 3** Change to the directory called ChatOps and use the command **python save_base_config.py** to save a copy of the configuration file to the CONFIG_FILES folder. Use **ls** and **more** to verify and view the configuration file.

**Step 4** Change to the directory called ChatOps and use the command **python create_incident.py** to send the change to Service Now and open a new incident.




**Step 5** Login to Service now with credentials provided by your instructor. Navigate to incidents and notice there is an incident with your name in it

**Optional Bonus Mission** (Difficulty Level – medium):

Modify the fields that are set when an incident is opened. Resources that you will need may include the Service Now API documentation.

**Step 6** Go back to the CSR1000V (out of guestshell) and add a loopback interface 0, assign an IP address of 7.7.7.7/32, add a description and save the configuraion

**Step 7** Go back the bash shell with the command **guestshell run bash**

**Step 8** Change to the directory called ChatOps and use the command **python config_change_incident.py** to send the change to Service Now and open a new incident.

**Step 9** Login to Service now with credentials provided by your instructor. Navigate to incidents and notice there is an incident with your name in it, also notice that the configuration changes are included in the incident.


**Optional Bonus Mission** (Difficulty Level - low)

Modify the script so that displays the changes and prompts you to open a ticket with service now or not.

# Common Task 5: Using EEM activating ChatOps

**Step 1** On the CSR1000V from privileged mode use **delete flash:/CONFIG_FILES/base-config**

**Step 2** Enter the following commands from configuration mode on the CSR1000V and save the configuration:

- **event manager applet config_change**
- **event syslog pattern "SYS-5-CONFIG_I" maxrun 120**
- **action 1.0 cli command "enable"**
- **action 2.0 command "guestshell run python /bootflash/ChatOps/config_change_approve.py"**
- **action 3.0 command "exit"**
- **action 4.0 command "end"**

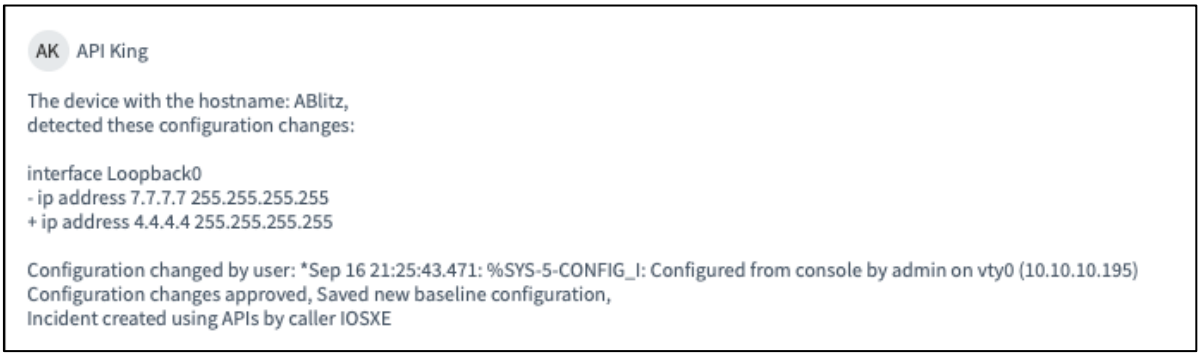
**Step 3** On the CSR1000V from privileged mode use **delete flash:/CONFIG_FILES/base-config**

**Step 4** Since we are on an SSH connection not a console, enable logging with the **terminal monitor** command

**Step 5** Go the bash shell with the command **guestshell run bash**

**Step 6** Run the save_base_conifg.py script to save a new base configuration. Execute with **python save_base_config.py**

**Step 7** Make a change to the Loopback 0 IP address and save the changes to kick off the script. Then look in Webex Teams for a new space called ChatOps. Enter **y** to accept the change.



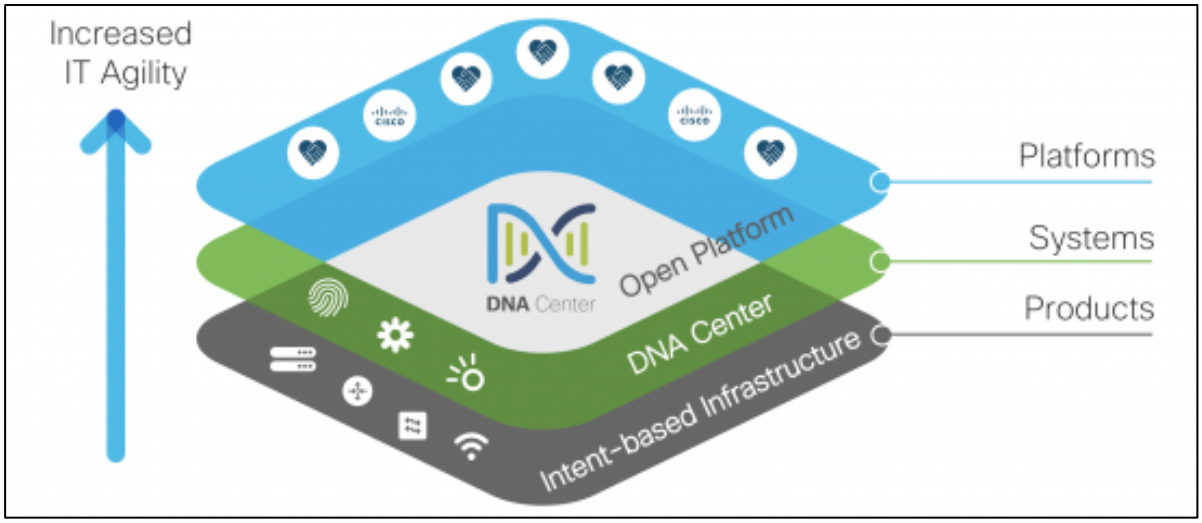

?

**Step 8** Test the script again by entering **n** this time.

**Step 9** Debug the script and test. Have fun!

**Step 10** **Disable EEM when done!**

# Lab 6: DNA Center Platform Introduction




## Introduction

This lab introduces you to some of the capabilities of DNA Center Platform via its API.

# Task 1: Explore DNA Center Platform API GUI

## Activity

Cisco DNA Center is at the heart of Cisco's intent-based network. DNA Center supports the expression of business intent for network use cases, such as base automation capabilities in the enterprise network. The Analytics and Assurance features of DNA Center provide end-to-end visibility into the network with full context through data and insights.

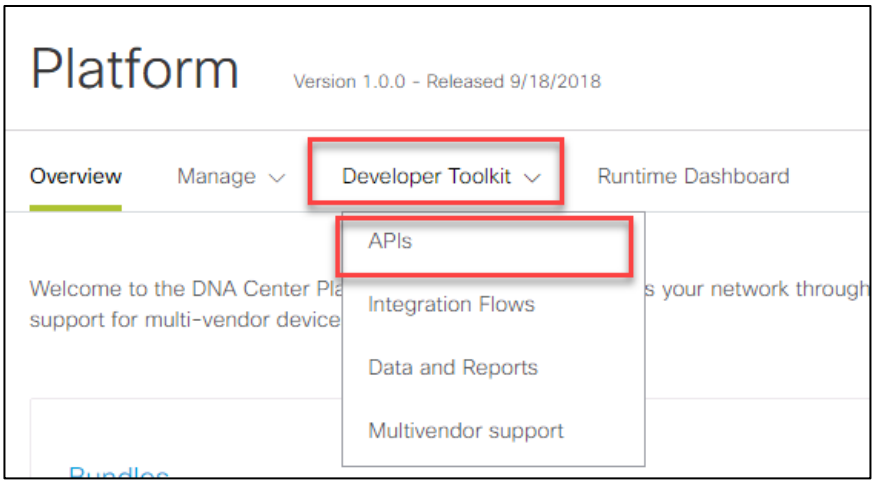
Cisco customers and partners can use DNA Center Platform to create value-added applications that leverage the native capabilities of DNA Center. You can use DNA Center Intent APIs, Integration Flows, Events and Notification Services, and the optional DNA Multivendor SDK to enhance the overall network experience by optimizing end-to-end IT processes, reducing Total Cost of Ownership (TCO), and developing new value-added networks.

This part of the lab will familiarize you with DNA Center's Platform API GUI.

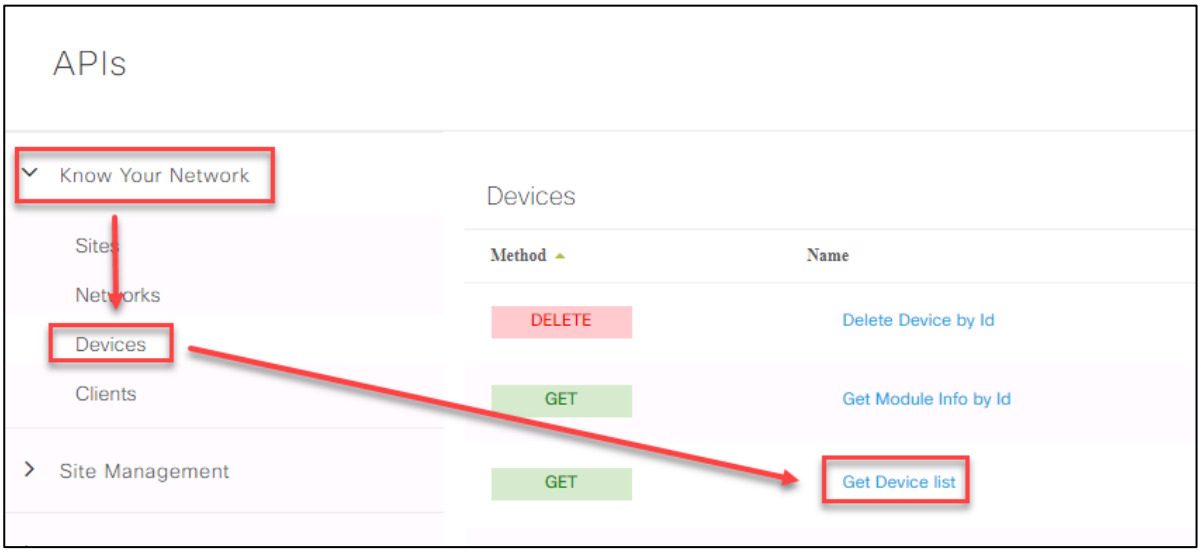
**Step 1** Log into DNA Center at https://10.10.10.204 with admin/ISEisC00L (those are zeros)

**Step 2** Click Platform at the top of the page.

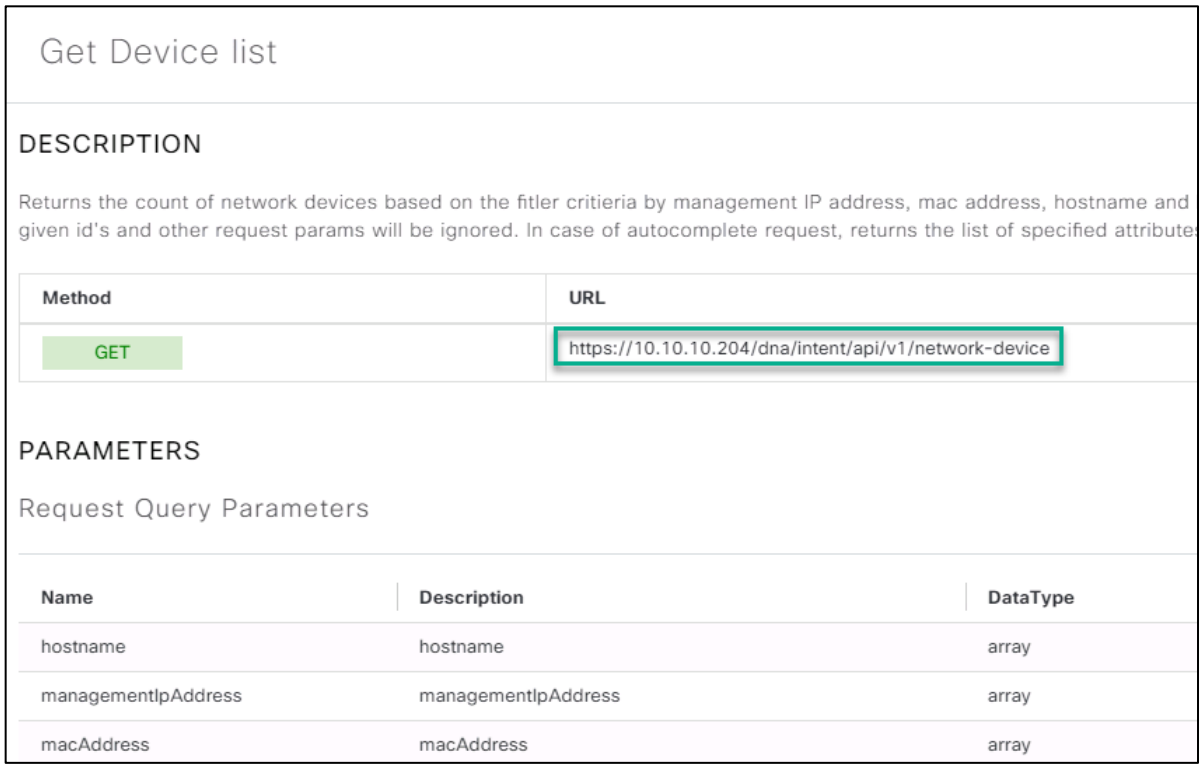

**Step 3** Select 'Developer Toolkit' and then 'APIs'



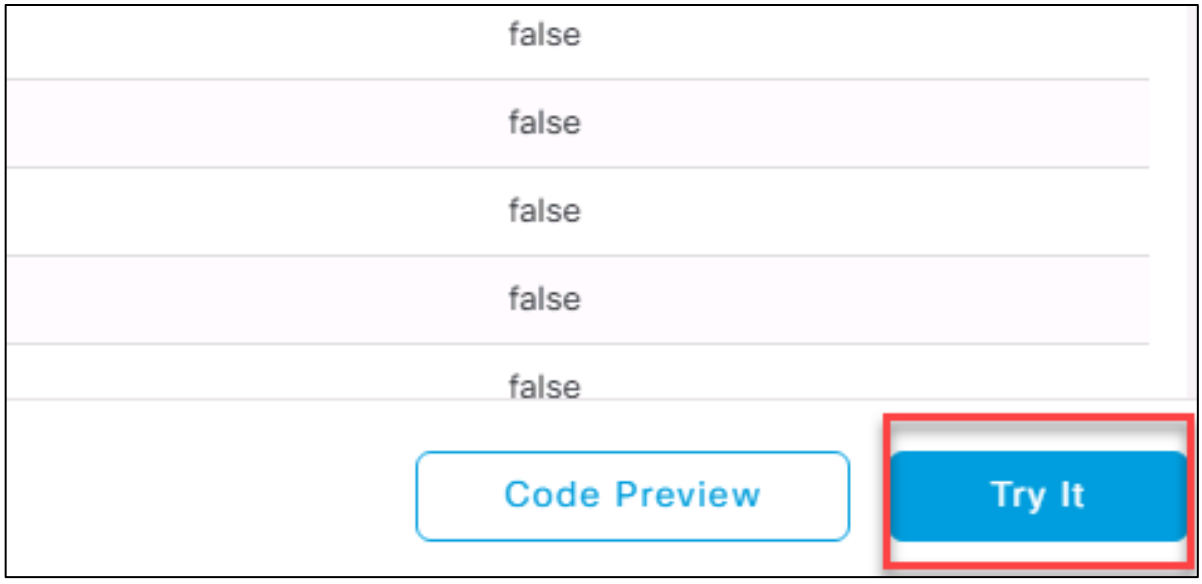

**Step 4** Navigate to "Get Device List" as Shown:



**Step 5** You screen should look similar to this:

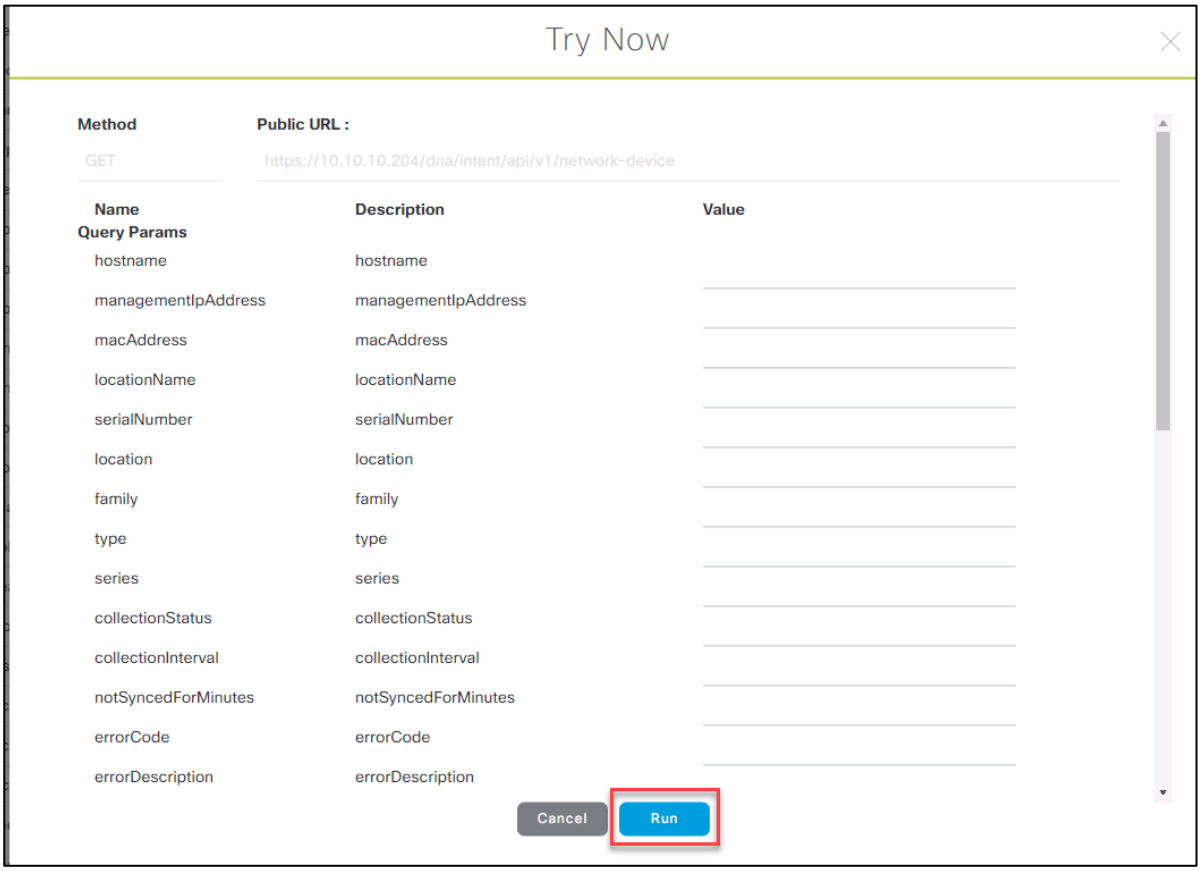



**Step 6** In the bottom right hand corner click "Try It"

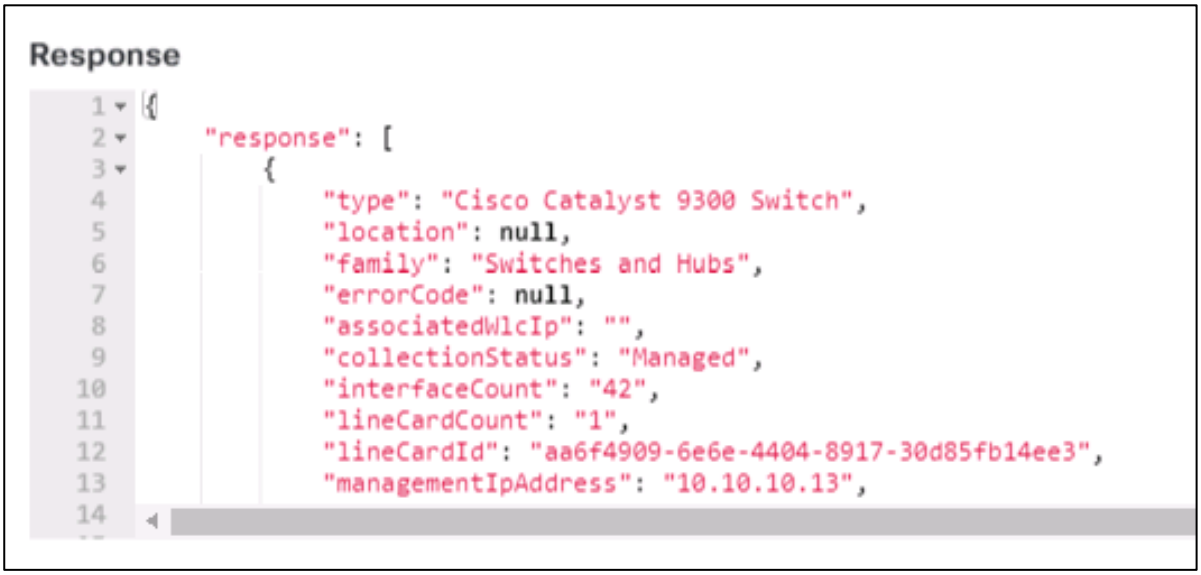



**Step 7** Click Run

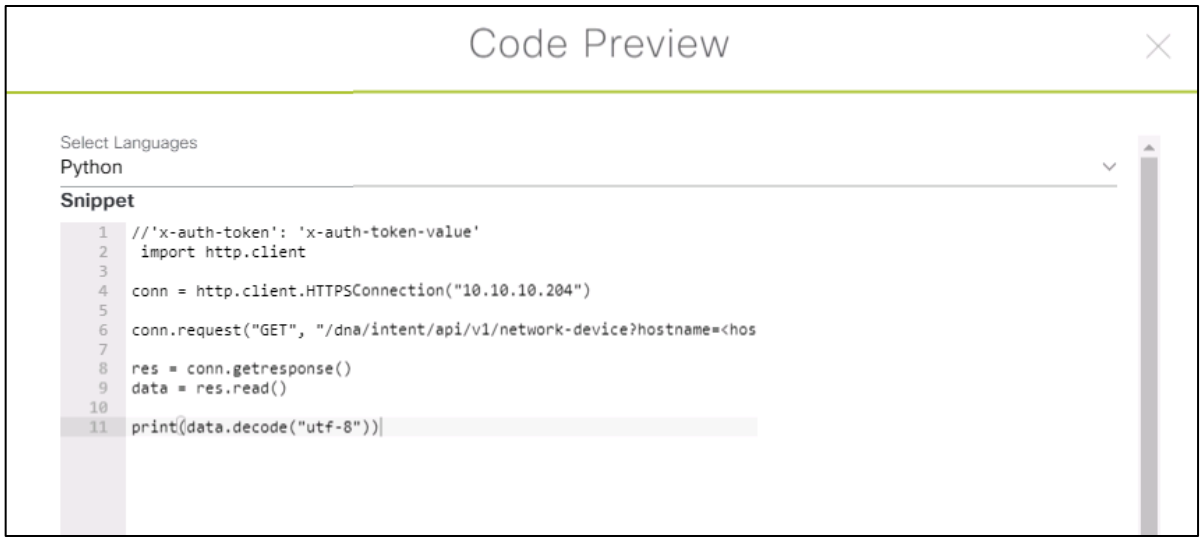



**Step 8** Scroll down and view the response. Then click Cancel.

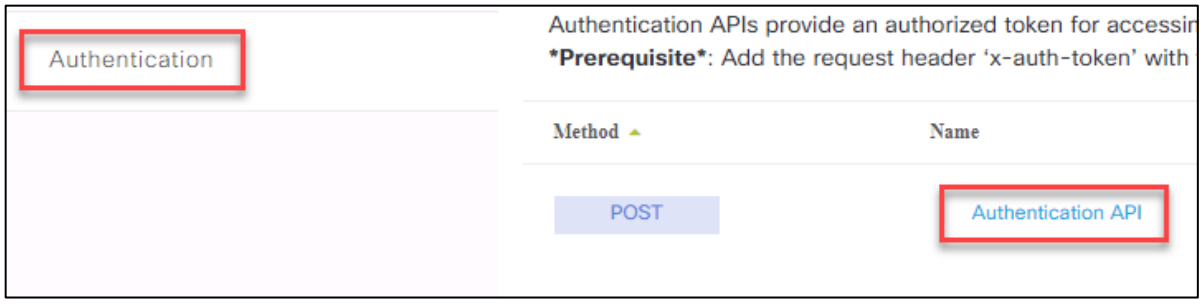



**Step 9** In the bottom right hand corner click “Code Preview”


**Step 10** Select Python as the language and observe the output. Then click Close. Then click the X to close the Devices window.

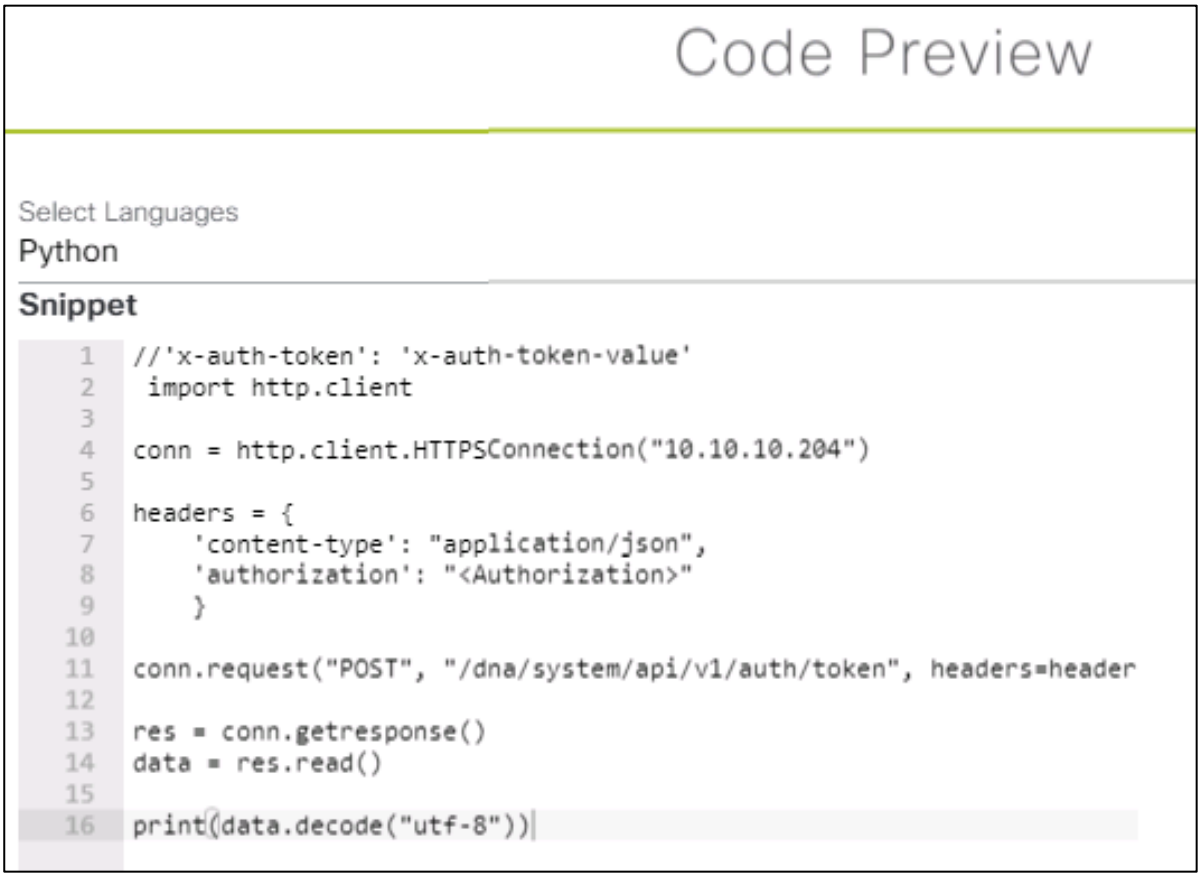



**Step 11** Select "Authentication" and then "Authentication API"



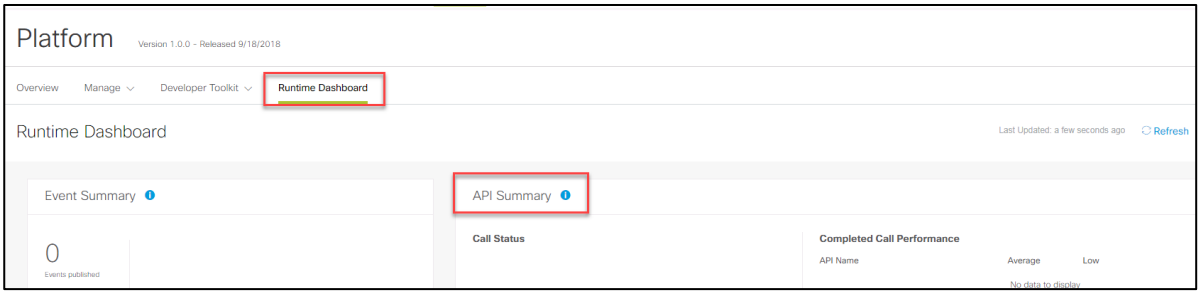

**Step 12** Note the Synax. Select "Code Preview" and set the language to "Python". Observe the results.




**Step 13** Close the authentication windows.

**Step 14** Select “Runtime Dashboard” and note the API Summary Section



# Lab 7: DNA Center w/Python

## Introduction

Through DNA Center API, you could use Python to create a script to automate DNA Center related activities. This Lab provides an approach to this capability.

## Advanced Path – Task 1 (Recommended)

**Step 1** Create a new python script that will connect to DNA Center and retrieve an access token. You will need to connect to https://10.10.10.204 with the credentials of admin/ISEisC00L (those are zeros)

## Advanced Path – Task 2 (Recommended)

**Step 1** Add the script you created to use the access token and fetch information about a device in DNA Center. The device is at IP address 10.10.10.100

## Optional Bonus Mission 1 (Difficulty Level – Medium)

- Use pprint or other tool to format the output to make it more human readable. See Foundation Task 2 for a possible solution.

## Optional Bonus Mission 2 (Difficulty Level – Medium)

- Use the BlitzAdvanced-master/dna_center_module/configuration_changes_monitoring.py script as a base to compare the old and new configuration file using DNA Center APIs

# Foundational Path – Task 1: Authenticating to DNA Center

## Activity

As before, with other APIs, you will need an Access Token in other to be authenticated and authorized in the DNA Center. You will be reviewing, but not modifying code.

Generating an Authorization Token

DNA Center accepts REST requests from authenticated users only. To authenticate to DNA center, you must submit your user credentials. Successful authentication returns an authorization token that you can use to issue subsequent requests.

**Step 1** Open from the BlitzAdvanced_Master folder on the desktop the file located at dna_center_module\lab_a_intro using Atom. This file contains a number of functions to simply the use of the DNA Center APIs.

**Step 2** Let Examine the beginning of the file

```
from requests.auth import HTTPBasicAuth  # for Basic Auth
from urllib3.exceptions import InsecureRequestWarning  # for insecure https warnings

from config import DNAC_URL, DNAC_PASS, DNAC_USER

urllib3.disable_warnings(InsecureRequestWarning)  # disable insecure https warnings

DNAC_AUTH = HTTPBasicAuth(DNAC_USER, DNAC_PASS)
```

**Step 3** The HTTPBasicAuth set authentication properties of username and password.

**Step 4** What is the purpose of the statement:

```
from config import DNAC_URL, DNAC_PASS, DNAC_USER
```

**Step 5** Open the file config.py and examine it, part of it should look something like:

```
# Update this section with the DNA Center server info and user information
DNAC_URL = 'https://172.28.97.216'
DNAC_USER = 'admin'
DNAC_PASS = 'Cisco123'
# For Classroom Use
#DNAC_URL = 'https://10.10.10.204'
#DNAC_USER = 'admin'
#DNAC_PASS = 'ISEisC00L'
```

**Step 6** This allows us to store our authentication credentials in a single file. Comment out the current credentials and uncomment out the credentials for 10.10.10.204

```
# Update this section with the DNA Center server info and user information
#DNAC_URL = 'https://172.28.97.216'
#DNAC_USER = 'admin'
#DNAC_PASS = 'Cisco123'
# For Classroom Use
DNAC_URL = 'https://10.10.10.204'
DNAC_USER = 'admin'
DNAC_PASS = 'ISEisC00L'
```

In the file dna_center_module\lab_a_intro the authentication portion is the function get_dnac_jwt_token

**Step 7** Execute the python script and observe the output.

```
def get_dnac_jwt_token(dnac_auth):
    """
    Create the authorization token required to access DNA C
    Call to DNA C - /api/system/v1/auth/login
    :param dnac_auth - DNA C Basic Auth string
    :return: DNA C JWT token
    """
    log("Executing Function: "+inspect.stack()[0][3])
    url = DNAC_URL + '/api/system/v1/auth/login'
    header = {'content-type': 'application/json'}
    response = requests.get(url, auth=dnac_auth, headers=header, verify=False)
    response_header = response.headers
    dnac_jwt_token = response_header['Set-Cookie']
    return dnac_jwt_token
```

**Step 8** The statements that establish the connection and create the token are:

```
# get the DNA Center JWT auth

dnac_jwt_auth = get_dnac_jwt_token(DNAC_AUTH)
print('The DNA Center Auth JWT is: ', dnac_jwt_auth)
```

# Foundational Path – Task 2: Retrieving Client information from DNA Center

## Activity

In this task, you will be adding code to the end of the lab_a_intro.py file that will allow us to gather information about a client.

**Step 1** Review the get_client_info function:

```
def get_client_info(client_ip, dnac_jwt_token):
    """
    This function will retrieve all the information from the client with the IP
    :param client_ip: client IPv4 address
    :param dnac_jwt_token: DNA C token
    :return: client info, or {None} if client does not found
    """
    log("Executing Function: "+inspect.stack()[0][3])
    url = DNAC_URL + '/api/v1/host?hostIp=' + client_ip
    header = {'content-type': 'application/json', 'Cookie': dnac_jwt_token}
    response = requests.get(url, headers=header, verify=False)
    client_json = response.json()
    try:
        client_info = client_json['response'][0]
        return client_info
    except:
        return None
```

**Step 2** Create a variable for the Client's IP Address and assign it the value of 10.10.10.100

**Step 3** We then want to call the get_client_info function and pass it the appropriate values. (Try to use Exception Handling to accomplish this).

## Optional Bonus Mission 1 (Difficulty Level – Medium)

- Use pprint or other tool to format the output to make it more human readable. See below for a possible solution.

### Possible Solution:

```
# get the client info

client_ip_add = '10.10.10.22'
client_detail = get_client_info(client_ip_add, dnac_jwt_auth)
print('The information for the client with the IP address: ', client_ip_add)
utils.pprint(client_detail)
```

```
C:\Users\student\BlitzAdvanced\dna_center_module>python lab_a_intro.py
https://10.10.10.204
Executing Function: get_dnac_jwt_token
The DNA Center Auth JWT is:  X-JWT-ACCESS-TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI1Yj
1ZDlhYjhiYWY0NDAwOGJmYWZkOTEiLCJhdXRoU291cmNlIjoiaW50ZXJuYWwiLCJ0ZW5hbnROYW1lIjoiVE5UMCIsInJvbGVzIj
bIjViMjVkOWFhOGJhZjQ0MDA4YmZhZmQ5MCJdLCJ0ZW5hbnRJZCI6IjViMjVkOWE5OGJhZjQ0MDA4YmZhZmQ4ZSIsImV4cCI6MTU
0MDE3NzU0NywidXNlcm5hbWUiOiJhZG1pbiJ9.Yn0W2h0hLfrtmDMAkViEH60_e2VJ_1_0bqB7pFHuq_c;Version=1;Comment=
;Domain=;Path=/;Max-Age=3600;Secure;HttpOnly
get_client_info
The information for the client with the IP address:  10.10.10.100
{
    "hostIp" : "10.10.10.100" ,
    "hostMac" : "00:50:56:a0:64:40" ,
    "hostName" : "ise" ,
    "hostType" : "wired" ,
    "connectedNetworkDeviceId" : "8302c1fb-88e8-4f12-899f-c6566650702e" ,
    "connectedNetworkDeviceIpAddress" : "10.10.10.22" ,
    "connectedInterfaceId" : "0e017cb3-16de-4ca9-a29c-59a36e9d1ec9" ,
    "connectedInterfaceName" : "GigabitEthernet1/0/13" ,
    "connectedNetworkDeviceName" : "POD21-SW2-BORDER.isesda.com" ,
    "vlanId" : "1" ,
    "lastUpdated" : "1537865287666" ,
    "source" : "200" ,
    "subType" : "UNKNOWN" ,
    "accessVLANId" : "1" ,
    "id" : "c7e14bbd-90e9-422d-902d-6067affbe61f"
}
```

# Optional Bonus Mission 2 (Difficulty Level – Medium)

**Step 1** Examine the file: BlitzAdvanced-master/dna_center_module/configuration_changes_monitoring.py

**Step 2** Note the configuration difference section

```
def compare_configs(cfg1, cfg2):
    """
    This function, using the unified diff function, will compare two config files and identify the changes.
    '+' or '-' will be prepended in front of the lines with changes
    :param cfg1: old configuration file path and filename
    :param cfg2: new configuration file path and filename
    :return: text with the configuration lines that changed. The return will include the configuration for the sections
    that include the changes
    """

    # open the old and new configuration fiels
    f1 = open(cfg1, 'r')
    old_cfg = f1.readlines()
    f1.close

    f2 = open(cfg2, 'r')
    new_cfg = f2.readlines()
    f2.close

    # compare the two specified config files {cfg1} and {cfg2}
    d = difflib.unified_diff(old_cfg, new_cfg, n=9)

    # create a diff_list that will include all the lines that changed
    # create a diff_output string that will collect the generator output from the unified_diff function
    diff_list = []
    diff_output = ''
```

**Step 3** DNAC connects, device info is retrieved and the ‘show run’ command is executed

```
1  98        dnac_token = dnac_apis.get_dnac_jwt_token(DNAC_AUTH)
   99        print('\nDNA C AUTH TOKEN: ', dnac_token, '\n')
2  104       all_devices_info = dnac_apis.get_all_device_info(dnac_token)
3  112           device_run_config = dnac_apis.get_output_command_runner('show running-config', device, dnac_token)
```

**Step 4** Note the service now import and incident creation

```
1   9    import dnac_apis
    10   import service_now_apis
    11   import os
    12   import os.path
2   153                  # create ServiceNow incident using ServiceNow APIs
    154                  incident = service_now_apis.create_incident(short_description, comment, SNOW_DEV, 3)
```

**Step 5** Run this, make changes as necessary for it to function.

# Lab 8: SD-WAN

In lab 8 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.

## Advanced Path – Task 1 (Recommended)

**Step 1** From your Student Desktop make a new python script and import the following:

- requests
- urllib3
- configparser

**Step 2** Define a function called **initialize_connection** that uses the following from a file called **package_config.ini** (already created and defined):

- **ipaddress (vManage)**
- **username**
- **password**

**Step 3** Enable the use of self-signed certificates

**Step 4** Make an API call to vManage that will return a list of the devices as raw json and print as output using the python script you built

## Advanced Path – Task 2 (Recommended)

**Step 1** Format the output using whatever method and tools you like to make the json more human friendly. See Foundation Task 2 below if you want a couple of examples.

**Step 2** Post your script to the classroom Webex Teams room so everyone can learn from your script

### Optional Bonus Mission (Difficulty Level – Medium)

- Post the list of devices to the classroom Webex Teams room, who will be the first?

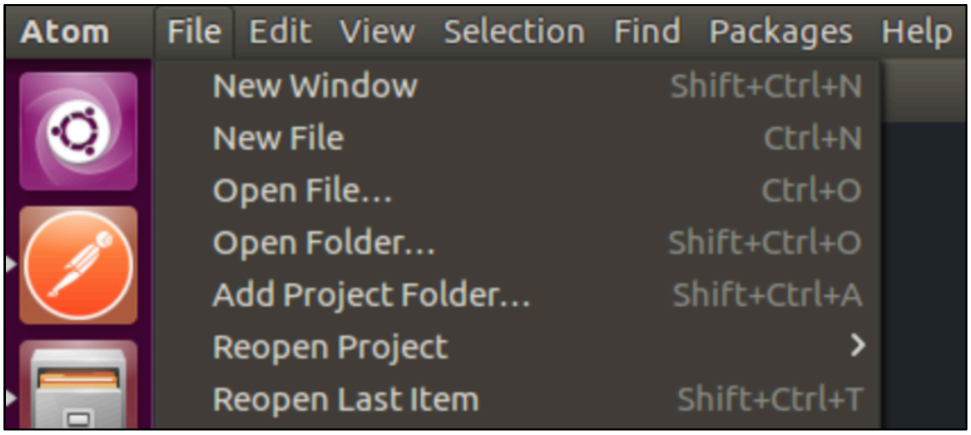
# Foundation Path – Task 1

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

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

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


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




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

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

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

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

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

Indent

```
def initalize_connection(ipaddress,username,password):

    """
    This function will initialize a connection to thevManage
    :param ipaddress: This is the IP Address and Port number of vManage
    :param username:  This is the username for vManage (admin in our lab)
    :param password:  This is a password for vManage (admin in our lab)
    These will be set in a file called package_config.ini
    """
```

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

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

Indent

```
def initalize_connection(ipaddress,username,password):

    """
    This function will initialize a connection to thevManage
    :param ipaddress: This is the IP Address and Port number of vManage
    :param username:  This is the username for vManage (admin in our lab)
    :param password:  This is a password for vManage (admin in our lab)
    These will be set in a file called package_config.ini
    """

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

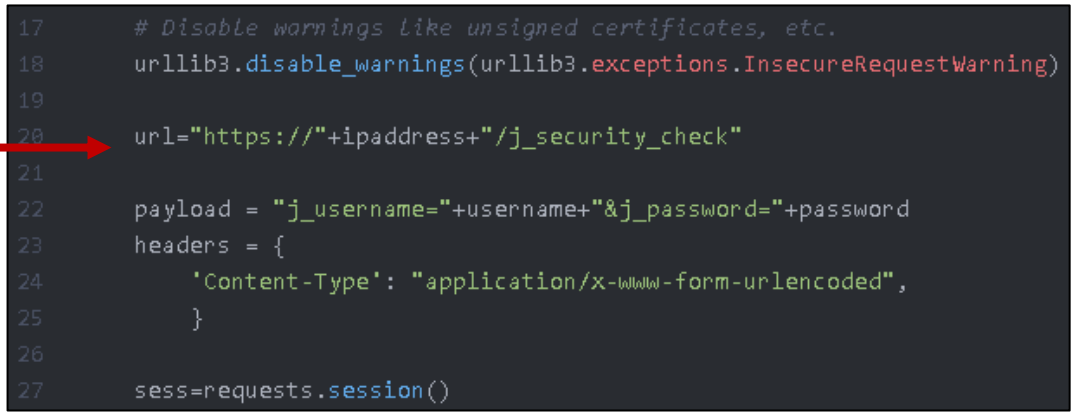
**Step 6** Next you will define where to authenticate and how to authenticate to the vManage. Add the following lines at the same indent level as the previous step:

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

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

sess=requests.session()
```

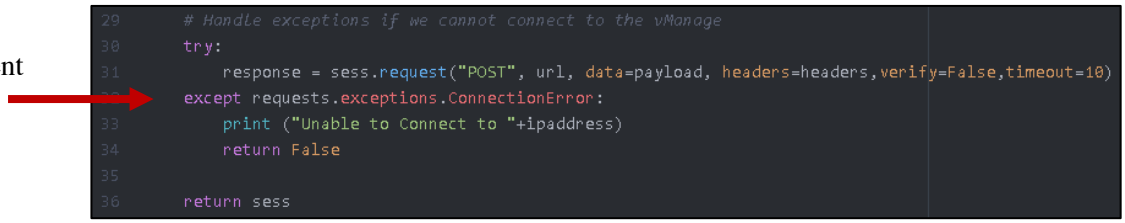
Indent




**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
```

Indent




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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Foundation Path – Task 2

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

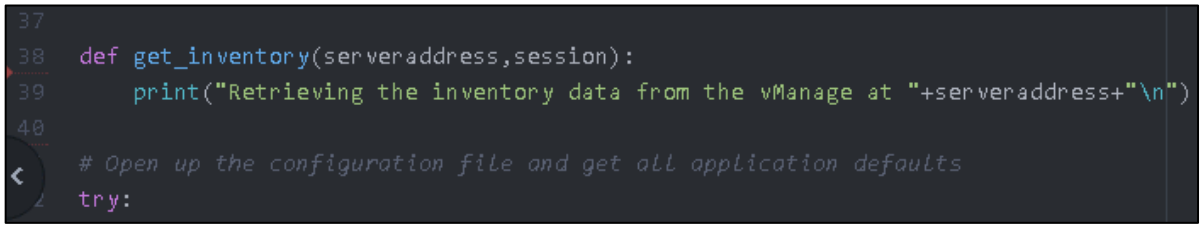
| Device Name | vManage GUI and REST API | Username | Password |
|---|---|---|---|
| vManage-1 | 1.4.28.28 | admin | admin |

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

**Step 2** Open the finished script from the previous task called **SDWAN.py**. This will be the starting point for this lab.

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

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




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

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


**Step 5** At the bottom of the script add an indented line
`get_inventory(serveraddress,session)`

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

**Step 6** Save the script.

**Step 7** Go to the Terminal and change to the SDWAN-Programmability directory with the command **cd SDWAN-Programmabiltiy** (if not already there) and execute the command **python SDWAN.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-IH4KQU5 MINGW64 /c/SDWAN-Programmability (master)
$ python PythonLab5.py
Viptela Configuration:
vManage Server Address: 1.4.28.28
vManage Username: admin
Successful you will issue API commands here in later labs
Retrieving the inventory data from the vManage at 1.4.28.28

{'header': {'generatedOn': 1555415811333, 'viewKeys': {'uniqueKey': ['system-ip'], 'preferenceKey':
'columns': [{'title': 'Hostname', 'property': 'host-name', 'display': 'iconAndText', 'iconProperty'
'hideable': False, 'icon': [{'key': 'vmanage', 'value': 'images/vmanage_table.png'}, {'key': 'vedge
es/vedge_table.png'}, {'key': 'vedge-vbond', 'value': 'images/vedge-vbond_table.png'}, {'key': 'vsm
mages/vsmart_table.png'}, {'key': 'vbond', 'value': 'images/vbond_table.png'}], 'width': 150, 'data
 {'title': 'State', 'property': 'state', 'display': 'iconAndToolTip', 'iconProperty': 'state', 'too
tate_description', 'defaultPropertyKey': 'reachability', 'defaultPropertyValue': 'reachable', 'icor
n', 'value': 'images/device_state_green.png'}, {'key': 'red', 'value': 'images/device_state_red.png
ow', 'value': 'images/device_state_yellow.png'}, {'key': 'default', 'value': 'images/device_state_d
th': 20, 'dataType': 'string'}, {'title': 'System IP', 'property': 'system-ip', 'hideable': False,
taType': 'ipv4'}, {'title': 'Reachability', 'property': 'reachability', 'display': 'multiColumns',
 'reachable', 'value': '616161', 'property': 'reachability'}, {'key': 'unreachable', 'value': 'ef5
'reachability'}, {'key': 'auth-failed', 'value': 'ffb300', 'property': 'reachability'}, {'key': 'st
```

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

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

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

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

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

{'deviceId': '172.1.0.18', 'system-ip': '172.1.0.18', 'host-name': 'vManage-1', 'reachability': 'reachable', 'status
': 'normal', 'personality': 'vmanage', 'device-type': 'vmanage', 'timezone': 'UTC', 'device-groups': ['"No groups"']
, 'lastupdated': 1555412615226, 'domain-id': '0', 'board-serial': '69DE12CE976BC070D5184152E0298516', 'certificate-v
alidity': 'Valid', 'max-controllers': '0', 'uuid': '618c3989-5dbe-4f92-95b8-cca81d1a466e', 'controlConnections': '11
', 'device-model': 'vmanage', 'version': '18.4.0', 'connectedVManages': ['"172.1.0.18"'], 'site-id': '1001', 'latitu
de': '37.666684', 'longitude': '-122.777023', 'isDeviceGeoData': False, 'platform': 'x86_64', 'uptime-date': 1555047
420000, 'statusOrder': 4, 'device-os': 'next', 'validity': 'valid', 'state': 'green', 'state_description': 'All daem
ons up', 'model_sku': 'None', 'local-system-ip': '172.1.0.18', 'total_cpu_count': '6', 'testbed_mode': False, 'layou
tLevel': 1}
======================
{'deviceId': '172.1.0.16', 'system-ip': '172.1.0.16', 'host-name': 'vSmart-1', 'reachability': 'reachable', 'status'
: 'normal', 'personality': 'vsmart', 'device-type': 'vsmart', 'timezone': 'UTC', 'device-groups': ['"No groups"'], '
lastupdated': 1555412683910, 'domain-id': '1', 'board-serial': '07F4E30F8E9368C48C066D1C5AB5198D', 'certificate-vali
dity': 'Valid', 'uuid': '6e4d0357-e883-4931-853e-25e5a85ad979', 'controlConnections': '13', 'device-model': 'vsmart'
, 'version': '18.4.0', 'connectedVManages': ['"172.1.0.18"'], 'site-id': '1011', 'ompPeers': '7 (10)', 'latitude': '
37.666684', 'longitude': '-122.777023', 'isDeviceGeoData': False, 'platform': 'x86_64', 'uptime-date': 1555057920000
, 'statusOrder': 4, 'device-os': 'next', 'validity': 'valid', 'state': 'green', 'state_description': 'All daemons up
', 'model_sku': 'None', 'local-system-ip': '172.1.0.16', 'total_cpu_count': '2', 'testbed_mode': False, 'layoutLevel
': 2}
======================
{'deviceId': '172.1.0.17', 'system-ip': '172.1.0.17', 'host-name': 'vSmart-2', 'reachability': 'reachable', 'status'
```

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

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

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

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

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

## Optional Bonus MISSION (difficulty Easy):

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

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

**Step 1** Lets take the data and print it in columns with some labels next. Navigate to the desktop the SDWAN folder and open the file name **PythonLab6Source.txt** with Atom and copy entire contents to your clipboard.

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

**Step 2** Paste the contents of the clipboard into the **PythonLab6.py script** that you are working on and place it below the previous lines you added. This section of code is yet another way to format the resulting json data and display it in a more friendly way.

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

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

**Step 3** 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   F0C2539C   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 4** End of Foundation Task

# Optional Bonus Mission 2 (Difficulty Level – Medium)

- Post the list of devices to the classroom Webex Teams room, who will be the first?

# Lab 9: WhatsOp Use Case

In lab 9 you will construct the WhatsOp use case which will integrate DNAC, CSR1000V, ServiceNow, and Webex Teams.

## Common Task 1: Setting up the CSR1000V for Guestshell (Skip Task 1 if previously completed in the ChatOps Use Case)

Connect to your assigned CSR1000V and configure AAA, a virtual port group for the Guestshell connectivity, NAT with the Guestshell, then enable guestshell and test connectivity.

## Activity

**Step 5** SSH to your assigned CSR1000v using the username **admin** and the password **ISEisC00L**

**Step 6** Configure IOX on from the configuration mode of the CSR1000V by entering the command **iox**.

**Step 7** Confirm that IOX is running by issuing the **show iox** command from privileged exec mode. This may take a minute for the services to show running.

```
ABlitz#show iox
Virtual Service Global State and Virtualization Limits:

Infrastructure version : 1.7
Total virtual services installed : 1
Total virtual services activated : 1

Machine types supported   : LXC
Machine types disabled    : KVM

Maximum VCPUs per virtual service : 0
Resource virtualization limits:
Name                         Quota     Committed     Available
------------------------------------------------------------------
system CPU (%)                  75             5            70
memory (MB)                   1024           512           512
bootflash (MB)               20000           756          4517


IOx Infrastructure Summary:
---------------------------
IOx service (CAF)    : Running
IOx service (HA)     : Not Running
IOx service (IOxman) : Running
Libvirtd             : Running
```

**Step 8** Configure the following AAA commands from configuration mode

- **aaa new-model**
- **aaa authentication login default local**
- **aaa authorization exec default local**

```
ABlitz(config)#aaa new-model
ABlitz(config)#aaa authentication login default local
ABlitz(config)#aaa new-model
```

**Step 9** Set a default route with **ip route 0.0.0.0 0.0.0.0 172.16.3.1**

**Step 10** Set a DNS server with the command **ip name-server 8.8.8.8**

```
ABlitz(config)#ip route 0.0.0.0 0.0.0.0 172.16.3.1
ABlitz(config)#ip name-server 8.8.8.8
```

**Step 11** a Virtual port group 0 with the command **interface VirtualPortGroup0**

**Step 12** Assign the following to the new Virtual Port Group

- **ip address 10.1.1.1 255.255.255.0**
- **ip nat inside**
- **description Used_for_GS_access**

```
ABlitz(config-if)#interface VirtualPortGroup 0
ABlitz(config-if)#ip add 10.1.1.1 255.255.255.0
ABlitz(config-if)#ip nat inside
ABlitz(config-if)#description Used_for_GS_Access
```

**Step 13** Set the outside NAT interface on Interface Gigabit Ethernet 2 with the command **ip nat outside** (the ip address will be unique to the pod)

```
interface GigabitEthernet2
 ip address 172.16.3.111 255.255.255.0
 ip nat outside
```

**Step 14** Configure the ACL that will be used for NAT with the commands

- **ip access-list extended Internet_ACL**
- **deny ip host 10.1.1.2 10.10.10.0 0.0.0.255**
- **permit ip host 10.1.1.2 any**

```
ip access-list extended Internet_ACL
 deny   ip host 10.1.1.2 10.10.10.0 0.0.0.255
 permit ip host 10.1.1.2 any
```

**Step 15** Next enter **ip nat inside source list Interne_ACL interface GigabitEthernet 2**

**Step 16** Configure the ACL that will be used for NAT with the commands

- **ip access-list extended Lab_ACL**
- **permit ip host 10.1.1.2 10.10.10.0 0.0.0.255**

```
ip access-list extended Lab_ACL
 permit ip host 10.1.1.2 10.10.10.0 0.0.0.255
```

**Step 17** Next enter the command **ip nat inside source list Lab_ACL interface GigabitEthernet 1**

**Step 18** Enter **app-hosting appid guestshell**

**Step 19** In the app sub configuration mode enter **vnic gateway1 virtualportgroup 0 guest-interface 0 guest-ipaddress 10.1.1.2 netmask 255.255.255.0 gateway 10.1.1.1 name-server 8.8.8.8**

**Step 20** Finally from privileged exec mode enter **guestshell enable** to start guestshell, be patient this may take a bit

**Step 21** Verify that the guestshell is started properly by issuing the **show app-hosting list** command

```
ABlitz#sho app-hosting detail
State                  : RUNNING
Author                 : Cisco Systems
Application
  Type                 : lxc
  App id               : guestshell
  Name                 : GuestShell
  Version              : 2.1(0.1)
Activated profile name : custom
  Description          : Cisco Systems Guest Shell XE for x86
Resource reservation
  Memory               : 512 MB
  Disk                 : 1 MB
  CPU                  : 799 units
  VCPU                 : 1
```

**Step 22** Verify connectivity is configured correctly by running **guestshell run sudo ping www.cisco.com**

**Step 23** Verfify connectivity with our DNA Center by using **guestshell run sudo ping 10.10.10.204**

# Task 2: Guest Shell Configuration (Portions may be skipped if ChatOps was configured)

Configure the GuestShell with the required packages and applications

## Activity

In this activity you will complete the necessary steps to have Guest Shell enable on the networking device.

**Step 1** From your CSR1000V start as bash shell in guestshell by running the command **guestshell run bash**

**Step 2** From the bash shell load the following packages: (The first 4 commands if done in the Chatops Use Case may be skipped)

- **sudo pip install –upgrade pip**
- **sudo pip install requests**
- **sudo pip install requests[security]**
- **sudo pip install ncclient==0.6.3**
- **sudo pip install pubnub**

**Step 3** Install and configure git (you will need a Github username and password) with the following commands: (skip this section if aleady done in the ChatOps User Case)

- **sudo yum install git -y**
- **git config --global user.name "your Github username" /Users/jeddemeule/Dropbox/Adv. Program Blitz/Sprint 2/WhatsOp/subscriber_listener.py**
- **git config --global user.email "youremail@yourdomain.com"**
- **git config --global credential.helper "cache -timeout 3600"**
- **git config --list**

**Step 4** Install some additional packages with the following commands: (skip this section if aleady done in the ChatOps User Case)

- **sudo yum install curl-devel -y**
- **sudo yum update -y nss curl libcurl**
- **sudo yum install nano -y**

**Step 5** Change to the bootflash directory with by entering **cd /bootflash**

**Step 6** Load the class files for this demo from github by using the command **git clone 'https://github.com/deplorablejed/Adv-Prog-Blitz/WhatsOp**

**Step 7** Verify the files replicated correctly by entering the command **ls /bootflash/WhatsOp**

# Common Task 3: Set WhatsOp Variables

## Activity

Configure the required variables in the config.py script using the Guestshell nano editor

**Step 8** From your Student Desktop get your personal Webex Teams token by going to http://developer.webex.com and then navigating to Documentation → Getting Started → Copy your token to your clipboard.

**Step 9** From your CSR1000V start as bash shell in guestshell by running the command **guestshell run bash**

**Step 10** Change to the /bootflash/ChatOps directory and use the command **nano config.py**. You should now be able to edit the variables in this file. Confirm the settings and if necessary set them as follows:

```
PUB_KEY = 'pub-c-INSTRCUTOR PROVIDED'

SUB_KEY = 'sub-c-INSTRUCTOR PROVIDED'

SEC_KEY = 'sec-c-INSTRUCTOR PROVIDED'

CHANNEL = 'IOS_XE_PUBNUB'


# Update this section with the Webex Teams token for each student

WEBEX_TEAMS_URL = 'https://api.ciscospark.com/v1'

WEBEX_TEAMS_AUTH = 'Bearer ' + 'YOUR TEAMS TOKEN HERE'

WEBEX_TEAMS_SPACE = 'Programmability Blitz Event Space'  # to be used when new
         instance created in ServiceNow


# Update this section with the DNA Center server info and user information

DNAC_URL = 'https://10.10.10.204'

DNAC_USER = 'admin'

DNAC_PASS = 'ISEisC00L1'


# Update this section with the Service Now instance to be used for labs

SNOW_URL = 'https://devXXXXX.service-now.com/api/now'

SNOW_ADMIN = 'IOSXE'
```

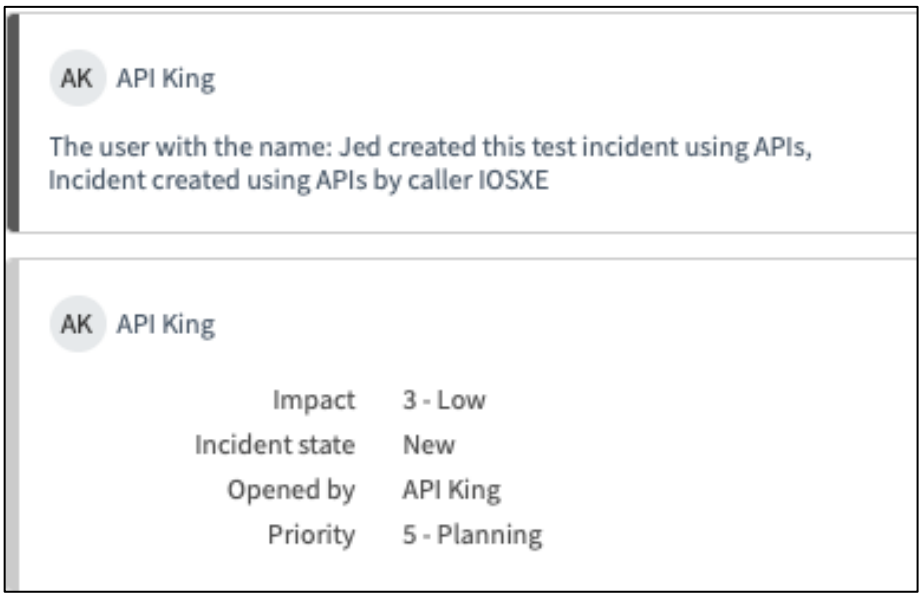
```
SNOW_DEV = 'IOSXE'

SNOW_PASS = 'IOSXE'

SNOW_INSTANCE = 'devXXXXX'


# Update this section with the info for the CSR1000V to be used during the IOS XE
        Module

IOS_XE_HOST = '10.1.1.1'

IOS_XE_USER = 'cisco'

IOS_XE_PASS = 'cisco'

IOS_XE_PORT = 830


UNAUTH_USER = 'blitz'

FOLDER_NAME = '/bootflash/WhatsOp
```

```
# Update this section with your PubNub keys
PUB_KEY = 'pub-c-9f62f33e-5fd4-4001-8755-bff7059bae5d'
SUB_KEY = 'sub-c-ebb3bb0e-d346-11e9-be73-4696cde2dee0'
SEC_KEY = 'sec-c-MDFlM2RkZGMtM2U2YS00MzAzLTg4MzItZGI2NDE3NmM1NzRl'
CHANNEL = 'IOS_XE_PUBNUB'


# Update this section with the Webeex Teams token for each student
WEBEX_TEAMS_URL = 'https://api.ciscospark.com/v1'
WEBEX_TEAMS_AUTH = 'Bearer ' + 'ZGNkYzgyMmUtOWM1Ny00YTFlLTgyZDItY2EzY2Y2NDgyYjkyYTZiNWI3NDEtYmEx_PF84_1eb65fdf
WEBEX_TEAMS_SPACE = 'Programmability Blitz Event Space'  # to be used when new instance created in ServiceNow


# Update this section with the DNA Center server info and user information
DNAC_URL = 'https://10.10.10.204'
DNAC_USER = 'admin'
DNAC_PASS = 'ISEisC00L1'

# Update this section with the Service Now instance to be used for labs
SNOW_URL = 'https://dev64635.service-now.com/api/now'
SNOW_ADMIN = 'IOSXE'
SNOW_DEV = 'IOSXE'
SNOW_PASS = 'IOSXE'
SNOW_INSTANCE = 'dev64635'


# Update this section with the info for the CSR1000V to be used during the IOS XE Module
IOS_XE_HOST = '10.1.1.1'
IOS_XE_USER = 'cisco'
IOS_XE_PASS = 'cisco'
IOS_XE_PORT = 830

UNAUTH_USER = 'blitz'

FOLDER_NAME = '/bootflash/WhatsOp'
```

# Common Task 4: Run Apps on the Guestshell

**Step 1** From your CSR1000V start as bash shell in guestshell by running the command **guestshell run bash**

**Step 2** Change to the directory **/bootflash/WhatsOp** and use the command **python create_incident.py** to send the change to Service Now and open a new incident.



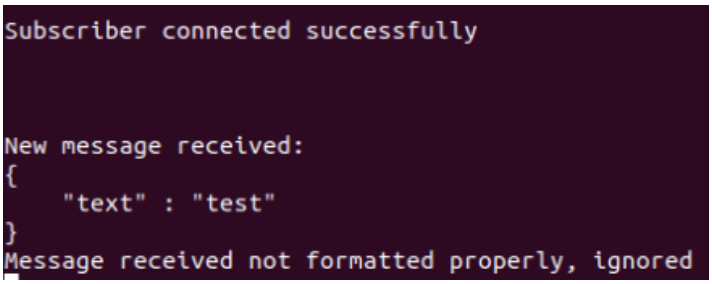

**Step 3** Login to Service now with credentials provided by your instructor. Navigate to incidents and notice there is an incident with your name in it

**Step 4** Enter the command **python subscriber_listener.py** this will show it connects successfully and then will wait for messages from PubNub

```
The device hostname: ABlitz1
https://10.10.10.204/dna/system/api/v1/auth/token
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiI1Y2Y3YThmNGNhOGI0ODAwYmVjMmYzZmIiLCJhdXRoU291cmNlIjoiaW50ZXJuYWwiLCJ0ZW5
hbnROYW1lIjoiVE5UMCIsInJvbGVzIjpbIjVjZjdhOGY0Y2E4YjQ4MDBiZWMyZjNmYSJdLCJ0ZW5hbnRJZCI6IjVjZjdhOGYzY2E4YjQ4MDBiZWMyZjNmOCI
sImV4cCI6MTU2OTI4MTYwMywiaWF0IjoxNTY5Mjc4MDAzLCJqdGkiOiJkYTJmMGEyNC1kZTY1LTQ3ODYtYjU5My02ZGM5YjczZDM4YjIiLCJ1c2VybmFtZSI
6ImFkbWluIn0.Zw127gm4sArfQ-qxsZAmsO1FVY4k9Tlw23tQ1BSHk41Ny0OAbqmAZlQqP7F22N0ejAUUHrdc9ePoin9RLOTLFTIX7Ur5XHGMl1_O4OtUVG0
xSW6o9B7p6Ek5ujtXRO0bO91vMc2nE_Ir3RIH9apLbgzO8q7TC_cZJZahblr6xHDJDebfizGgJoVqWZIjekVJ8NYQC-G_5HMy-aZalpfDMVph4EoCbbqC2qq
jg5gITuv4cVgRsRp69-3X_PKCni0J4QrzAEzZ5gWDGRtG6OWVy-gOE-JEhTXSA9Jrec7o5YoTrRSQu4ofiQ3OdN3kOeTrEy57U3yP-a2ylfinoQyfJg
ABlitz1.ablitz.com
200

Device Location: Global/Belgium/Brussels/Pod22-B1/Floor1

Subscriber connected successfully
```

**Step 5** Tell you instructor when your subscriber is listening and have them send you a test message from PubNub



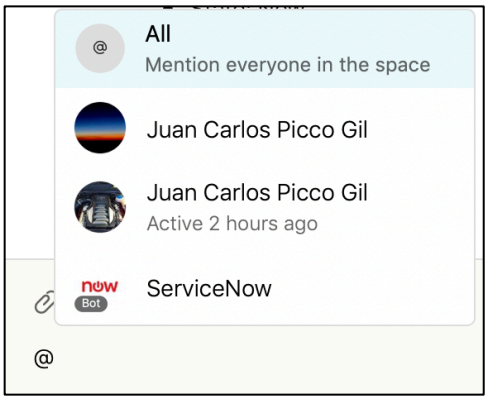

| | |
|---|---|
| **Note** | Notice that the screenshot above has a message that is not in the format that is expected. This subscriber_listener.py expects specific formatting and elements to be present. We will do this in a later task. |

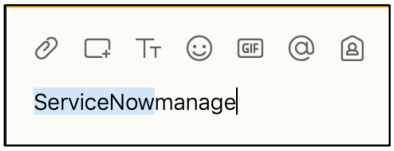
# Common Task 5: Configure the Webex Bot (Instructor Demo)

**Step 1** Go to the https://apphub.webex.com/teams/applications/servicenow-3117 and press **Add to Space**

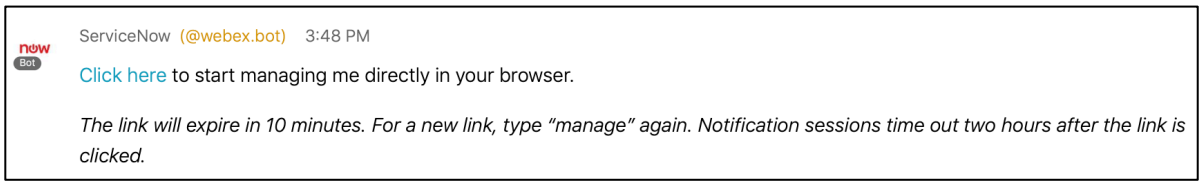
**Step 2** Choose the Classroom Space with all students added to the space

**Step 3** From the space type @ServiceNowmanage

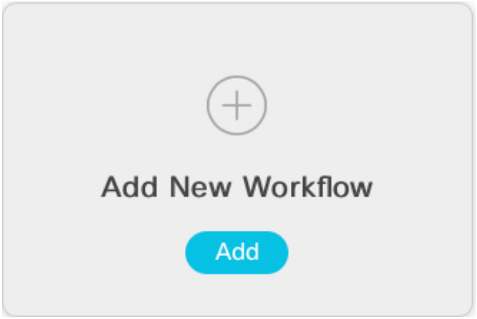



**Step 4** A link will be sent to you in a new ServiceNow space, select the link and walk thru the integration




**Step 5** Select the **Click here** link to open the ServiceNow Bot Workflows page. Then select **Add** on Add New Workflow




**Step 6** Give your workflow the following settings:

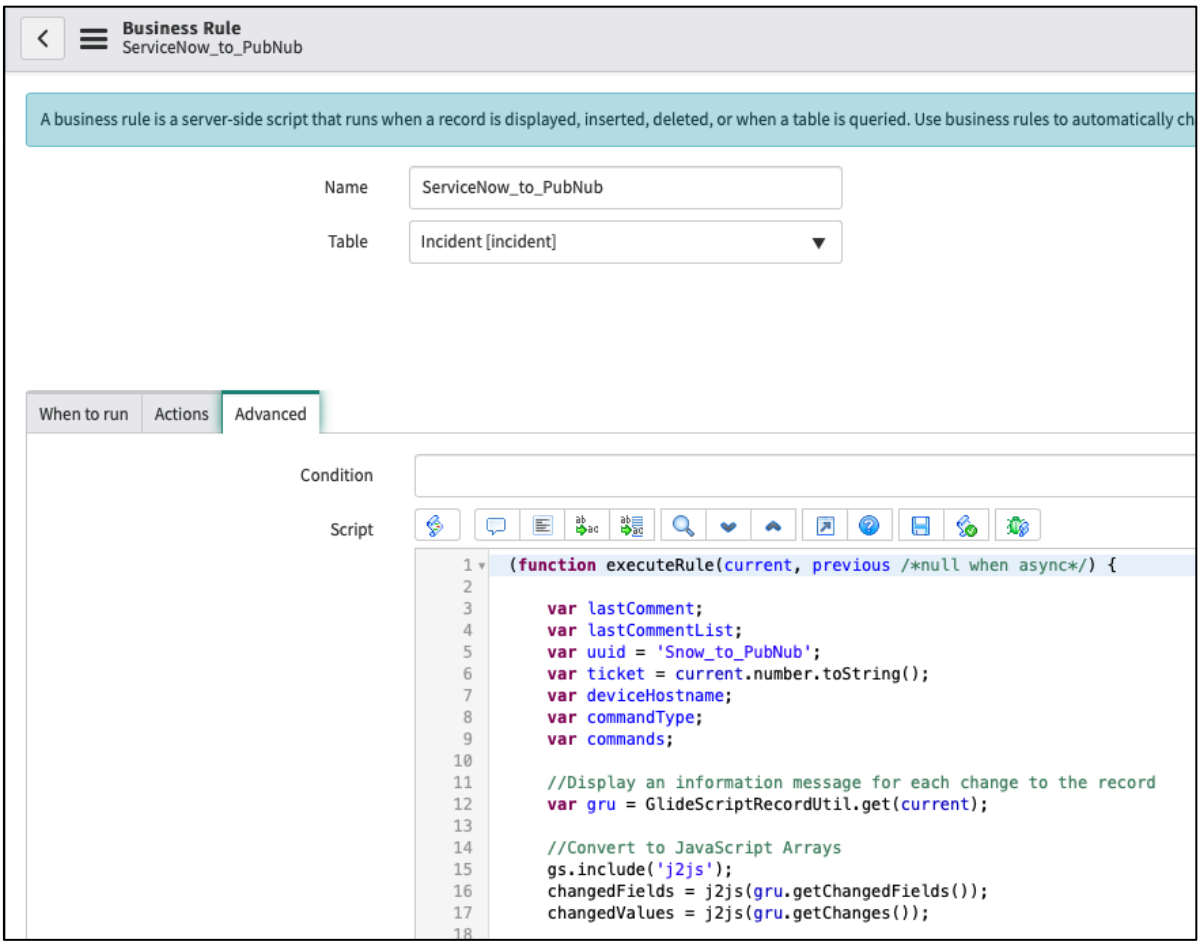
- Label: **ServiceNow**
- Version: **Madrid**

- Table: **incident**

**Step 7** Copy the entire script from Step C on the web page to your clipboard and them press **Save** to save the workflow.

**Step 8** Next paste the script on your clipboard into the **ServiceNow_to_Pubnub** Business rule on the **Advanced** tab in ServiceNow.




**Step 9** Open the incident and verify it was created by the script

# Common Task 6: Configure Monitored Routes on the CSR1000V

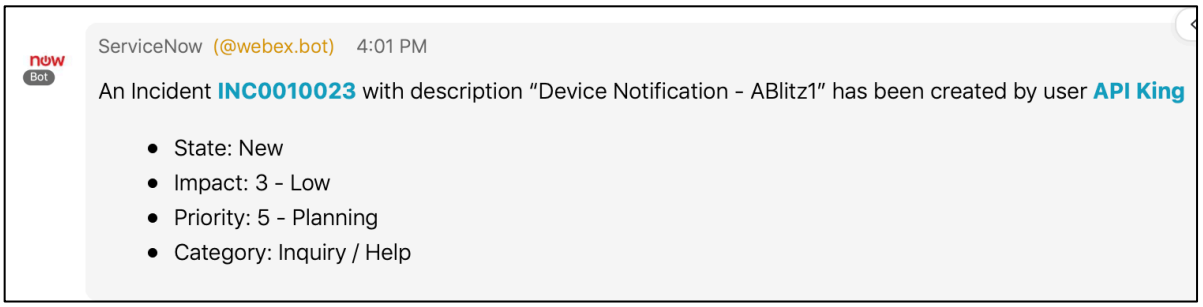
**Step 1** Open a second ssh session from your Ubuntu desktop to your CSR1000V.

**Step 2** From your CSR1000V run the command **show ip route** and verify that network 11.1.0.0 is present in the routing table.

**Step 3** Enter the command **guestshell run python /bootflash/WhatsOp/create_incident.py** to open a new incident on ServiceNow.

```
ABlitz1#guestshell run python /bootflash/WhatsOp/create_incident.py
/usr/lib64/python2.7/site-packages/cryptography/hazmat/primitives/constant_time.
py:26: CryptographyDeprecationWarning: Support for your Python version is deprec
ated. The next version of cryptography will remove support. Please upgrade to a
release (2.7.7+) that supports hmac.compare_digest as soon as possible.
  utils.PersistentlyDeprecated2018,

APIUSER ServiceNow sysid is: 02826bf03710200044e0bfc8bcbe5d3f

ServiceNow REST API call response: 201
('Created ServiceNow Incident with the number: ', 'INC0010023')

End Application Run

ABlitz1#
```

**Step 4** Notice that you have a new incident that now shows up in the Webex Teams Space. You may see other students incidents appearing in this space as well. You can find yours by looking at the output on the router, for example in the above screenshot you can see the incident number is **INC0010023**. Simply click the link INC0010023 to open the incident in ServiceNow




**Optional Bonus Mission** (Difficulty Level – medium):

Modify the fields that are set when an incident is opened.

# Common Task 7: Using EEM activating WhatsOp

**Step 1** On the CSR1000V from privileged mode use **delete flash:/CONFIG_FILES/base-config**

**Step 2** Enter the following commands from configuration mode on the CSR1000V and save the configuration:

- **event manager applet monitor_route**
- **event routing network 11.1.0.0/16 type remove ge 16 le 16 vrf default maxrun 30**
- **action 1.0 cli command "enable"**
- **action 2.0 cli command "guestshell run python /bootflash/WhatsOp/monitor_route.py 11.1.0.0/16"**
- **action 3.0 cli command "exit"**

- **action 4.0 cli command "end"**

**Step 3** On the CSR1000V from privileged mode go to **interface Vasileft 11** and enter **shutdown** to invoke EEM to trigger

**Step 4** Notice in Webex Teams you should see the incident and can open it up

**Note** The formatting you send to Guestshell via the ServiceNow Work Notes field must have very specific formatting as show below.

**Step 5** In the incident in ServiceNow enter the following text in the **Work Notes** field and press the **POST** button.

```
device:ABlitzX.ablitz.com (where X is your pod number)
config_type:exec
commands:show ip route
```

**Step 6** Note the routing table is displayed within the ServiceNow Incident

**Step 7** Proceed to troubleshoot the issue with any show commands that you find relavent

**Step 8** When you are ready to solve the problem enter the following in the **Work Notes** field and press **Post**

```
device:ABlitzX.ablitz.com (where X is your pod number)
config_type:config
commands:interface vasileft 11 | no shutdown
```

**Step 9** Debug the script if necessary. Have fun!