PyPlus - Class 1 - Git fundamentals

First create a GitHub account. Install the git application on Mac or Linux whichever system you are using. For the installation of git, refer to the Learning Python course Notes.

|  |
| --- |
| 1. Create a GitHub account  2. Create a new repository in GitHub  3. Clone the repository that you just created on GitHub into your home directory in the lab environment using git clone “url”  For example:  4. Configure your name and email address on the lab server:  $ git config --global user.name "John Doe"  $ git config --global user.email jdoe@domain.com  4. On your lab environment, go to repository and do “git init”  5. Check if the initialisation is ok, do “git status”  6. Git has 3 environments  - Actual Repository (.git)  - staging  - Working directory  Basically, there is a 2 step process, first we work on the files in the working directory, and then we use the command “git add” and “git rm” to put them in the staging environment and we move those files to the Actual Repository using the commands “git commit”. You can add files and folders using the “git add”. Using the “git status” you can verify on which environment those files are placed. If they on the working directory “the comment will be “*Untracked files:*  *(use "git add <file>..." to include in what will be committed)”.* After adding the files to the staging using the “git add” and now the “git status” will show the comments as  *“Changes to be committed:*  *(use "git restore --staged <file>..." to unstage)”* |

First create a working directory and create the files under it.

Pyplus - Class1 - Netmiko Overview(Part1)

lundi, 28 janvier 2019

15:20

Netmiko, in general, is trying to simplify the SSH connection and management process across a broad set of networking devices.

There's something like over 40+ device types supported in Netmiko, so various vendors and various levels of support,but there's over 40 of them supported in Netmiko.

Some general characteristics of Netmiko. Netmiko tries to set up the SSH connection in a way that's usable for automation, it tries to disable the output paging.

For example, if it's prompting for more, or ‘do you want to continue?’ on when you execute show commands, it's going to try to have that be disabled, so you get the entire show output.

It's also going to do other things with the terminal to try to make sure that it's working in a proper way.

When you execute show commands it's going to try to only get the show output, so it'll try to strip off the command echo. It'll try to strip off the trailing router prompt. It'll try to capture only the show output.

Similarly, in config mode, it tries to automatically enter you into config mode. It tries to automatically cause you to exit config mode.

One thing that Netmiko generally doesn't do is it doesn't understand the commands you are sending down the channel. If you send a wrong command to the router, netmiko will not correct it for you.If you send down ‘show IP’ and brief down the SSH channel and you put the command wrong, you typo it, Netmiko's not going to do anything about that. Netmiko is not going to say, ‘This is invalid’. What Netmiko will give you is literally the output of the router that comes back from that command.

In Cisco, if you get something like percent invalid command or something of that form,

Netmiko is just going to send you straight back that output

1. **To check the python version**

(py3\_venv) [sethuvignesh@ip-172-30-0-144 ~]$ python

Python 3.6.7 (default, Dec 21 2018, 20:31:01)

1. **To install netmiko**

(py3\_venv) [sethuvignesh@ip-172-30-0-144 ~]$ pip install netmiko

Collecting netmiko

Installing collected packages: netmiko

Successfully installed netmiko-2.3.0

1. **To import ConnectHandler module and getpass module**

from netmiko import ConnectHandler

from getpass import getpass

1. **To test if netmiko can connect to a device, here is the script**

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ vi netmiko\_overiew.py

from netmiko import ConnectHandler

from getpass import getpass

connect\_rtr3 = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = getpass(),

device\_type = 'cisco\_ios',

)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_overiew.py

Password:

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$

1. **To see if have logged in correctly to the correct device you need to**

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ vi netmiko\_overiew.py

from netmiko import ConnectHandler

from getpass import getpass

connect\_rtr3 = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = getpass(),

device\_type = 'cisco\_ios',

)

print(connect\_rtr3.find\_prompt())

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_overiew.py

Password:

cisco3#

1. **If you would need to see what netmiko does after connecting to the device**

from netmiko import ConnectHandler

from getpass import getpass

connect\_rtr3 = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = getpass(),

device\_type = 'cisco\_ios',

session\_log = 'my\_session.txt'

)

print(connect\_rtr3.find\_prompt())

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ ls

my\_session.txt

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat my\_session.txt

cisco3#

cisco3#terminal length 0

cisco3#terminal width 511

cisco3#

cisco3#(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$

cisco3#(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$

First, you need to know the correct device types. well how do I know the correct device types? If you don't know what device type to use, you can just put an invalid device type in and then Netmiko will give you all the valid device types, at least for the SSH context. For ‘telnet’, it's a little bit trickier. You probably have to look into the ‘ssh.dispatcher.py’ file to see the ‘telnet’ device types, but for SSH, it'll just straight give them to you if you put in an invalid device type. One of these ones we see up here is ‘cisco\_ios’. I'm going to go ahead and use a device type of ‘cisco\_ios’. Netmiko needs to know what the platform is that you're connecting to.

General form of Netmiko is ‘vendor\_network operating system’ and then there's optionally transport, so SSH, telnet, or potentially serial. Now if you don't specify the transport, then it's going to default to SSH.

Here is an example, I am using the invalid device\_type and you can see what netmiko does.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat netmiko\_overiew1.py

from netmiko import ConnectHandler

from getpass import getpass

connect\_rtr3 = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = getpass(),

device\_type = 'No idea'

)

print(connect\_rtr3.find\_prompt())

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_overiew1.py

Password:

Traceback (most recent call last):

File "netmiko\_overiew1.py", line 7, in

**device\_type = 'No idea'**

File "/home/sethuvignesh/VENV/py3\_venv/local/lib/python3.6/site-packages/netmiko/ssh\_dispatcher.py", line 216, in ConnectHandler

'**currently supported platforms are: {}'.format(platforms\_str))**

**ValueError: Unsupported device\_type: currently supported platforms are**:

a10

accedian

alcatel\_aos

alcatel\_sros

apresia\_aeos

arista\_eos

aruba\_os

avaya\_ers

avaya\_vsp

brocade\_fastiron

brocade\_netiron

brocade\_nos

brocade\_vdx

brocade\_vyos

calix\_b6

checkpoint\_gaia

ciena\_saos

cisco\_asa

cisco\_ios

cisco\_nxos

cisco\_s300

cisco\_tp

cisco\_wlc

cisco\_xe

cisco\_xr

coriant

dell\_dnos9

dell\_force10

dell\_isilon

dell\_os10

dell\_os6

dell\_os9

dell\_powerconnect

eltex

enterasys

extreme

extreme\_ers

extreme\_exos

extreme\_netiron

extreme\_nos

extreme\_slx

extreme\_vdx

extreme\_vsp

extreme\_wing

f5\_linux

f5\_ltm

f5\_tmsh

fortinet

generic\_termserver

hp\_comware

hp\_procurve

huawei

huawei\_vrpv8

ipinfusion\_ocnos

juniper

juniper\_junos

linux

mellanox

mrv\_optiswitch

netapp\_cdot

netscaler

ovs\_linux

paloalto\_panos

pluribus

quanta\_mesh

rad\_etx

ruckus\_fastiron

ubiquiti\_edge

ubiquiti\_edgeswitch

vyatta\_vyos

vyos

Pyplus - Class1 - Netmiko telnet only devices

vendredi, 8 mars 2019

19:44

If the cisco device do not support ssh, supports only telnet then code to login to the devices

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios\_telnet',

"fast\_cli": True

}

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_telnet\_only.py

Password:

cisco3#

(py3\_venv) [sethuvignesh@ip-172-30-0-144 ~]$ netstat -an | grep 23

tcp 0 0 172.30.0.144:48298 184.105.247.89:23 ESTABLISHED

Pyplus - Class1 - Netmiko send\_command()

vendredi, 22 février 2019

12:33

Earlier we were connecting in the following way. We were passing in our arguments, host, username, password, device type, session log. We were printing the prompt after we connected.

Now, I want to convert these arguments over to a device dictionary. In general, I like storing my devices in a dictionary format, as I think that makes it easier to manage a set of devices. I'm going to convert this all over to a dictionary, and then I'm going to pass the dictionary into Netmiko using the \*\* keyword arguments format.

Using \*\* keyword arguments, it is going to convert it from being a dictionary that it passes in, instead pass in all these key value pairs individually.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat netmiko\_ex1.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

    "host": 'cisco3.lasthop.io',

    "username": 'pyclass',

    "password": getpass(),

    "device\_type": 'cisco\_ios',

    #session\_log: 'my\_session.txt'

        }

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_ex1.py

Password:

cisco3#

Now what if we want to do a show command? We can do ‘connect\_rtr’, that's our Netmiko object, we can call the ‘send\_command()’ method. The ‘send\_command’ method is Netmiko's main way of executing show commands down the channel. I can do ‘send\_command(“show ip int brief”)’. I can print the output of that, and I can retrieve that output back from the channel.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat netmiko\_ex1.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

    "host": 'cisco3.lasthop.io',

    "username": 'pyclass',

    "password": getpass(),

    "device\_type": 'cisco\_ios',

    #session\_log: 'my\_session.txt'

        }

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.send\_command('sh ip int brief'))

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_ex1.py

Password:

Interface              IP-Address      OK? Method Status                Protocol

GigabitEthernet0/0/0   10.220.88.22    YES manual up                    up

GigabitEthernet0/0/1   unassigned      YES NVRAM  administratively down down

Gi0/0/1.146            unassigned      YES manual deleted               down

Gi0/0/1.147            unassigned      YES manual deleted               down

Gi0/0/1.148            unassigned      YES manual deleted               down

Gi0/0/1.149            unassigned      YES TFTP   deleted               down

Gi0/0/1.150            unassigned      YES manual deleted               down

GigabitEthernet0/1/0   unassigned      YES unset  down                  down

GigabitEthernet0/1/1   unassigned      YES unset  down                  down

GigabitEthernet0/1/2   unassigned      YES unset  down                  down

GigabitEthernet0/1/3   unassigned      YES unset  down                  down

Vlan1                  unassigned      YES unset  up                    down

Notice there's no command echo at the top. Notice there's no trailing router prompt. You've retrieved the information from the device. You can change the command, so if you want it to be a different command, and look at the output of that command.Now what Netmiko is doing behind the scenes to figure out if the ‘show\_command’ is done,because basically we have this issue with screen scraping, is we don't have a very good way of knowing that our command is done. How do we know that we've retrieved all the information that we need to retrieve? What Netmiko does is, it basically finds the current router prompt, it uses the current router prompt to determine that the current output is complete. Once it sees the current router prompt at the end of a given command, it'll know that ‘send\_command’ has completed, and it's retrieved everything it's supposed to retrieve. Also you can control this using this ‘expect\_string’ argument. If you want it to use a differentpattern, and not the trailing router prompt, to determine that a command is done, you can usethis ‘expect\_string’ argument.

connect\_rtr.send\_command('sh ip int brief', expect\_string=r'#')

Now ‘expect\_string’ does take regular expressions, so I'm passing in a Python raw string. It's basically escaping certain Python special characters so that I can pass it to the regular expression engine. Here I'm going to look for a trailing pound sign, which should be at the end of the router prompt.Then we're going to basically get our output back.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat netmiko\_ex1.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

    "host": 'cisco3.lasthop.io',

    "username": 'pyclass',

    "password": getpass(),

    "device\_type": 'cisco\_ios',

            }

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.send\_command('sh ip arp', expect\_string=r'#'))

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_ex1.py

Password:

Protocol  Address          Age (min)  Hardware Addr   Type   Interface

Internet  10.220.88.1            22   0062.ec29.70fe  ARPA   GigabitEthernet0/0/0

Internet  10.220.88.20          143   c89c.1dea.0eb6  ARPA   GigabitEthernet0/0/0

Internet  10.220.88.22            -   a093.5141.b780  ARPA   GigabitEthernet0/0/0

Internet  10.220.88.37          152   0001.00ff.0001  ARPA   GigabitEthernet0/0/0

Internet  10.220.88.38          101   0002.00ff.0001  ARPA   GigabitEthernet0/0/0

Pyplus - Class1 - Exercises

vendredi, 22 février 2019

12:35

1. In the lab environment use Netmiko to connect to one of the Cisco devices. Simply print the router prompt back from this device to verify you are connecting to the device properly.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat netmiko\_overiew.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

    "host": 'cisco3.lasthop.io',

    "username": 'pyclass',

    "password": getpass(),

    "device\_type": 'cisco\_ios',

        }

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_overiew.py

Password:

cisco3#

2. Connect to 2 NX-OS device in the lab, make sure you are using dictionaries to represent the two NX-OS devices. Additionally, use a for-loop to accomplish the Netmiko connection creation. Once again print the prompt back from the devices that you connected to.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat netmiko\_ex2.py

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

    device1 = {

     "host": "nxos1.lasthop.io",

     "username": "pyclass",

     "password": password,

     "device\_type": "cisco\_nxos",

               }

    device2 = {

     "host": "nxos2.lasthop.io",

     "username": "pyclass",

     "password": password,

     "device\_type": "cisco\_nxos",

              }

for device in (device1, device2):

     connect\_device = ConnectHandler(\*\*device)

     print(connect\_device.find\_prompt())

connect\_device.disconnect()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_ex2.py

Password:

nxos1#

nxos2#

3. For one of the Cisco IOS devices, use Netmiko and the send\_command() method to retrieve 'show version'. Save this output to a file in the current working directory.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat netmiko\_ex3.py

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

device1 = {

    "host": "cisco3.lasthop.io",

    "username": "pyclass",

    "password": password,

    "device\_type": "cisco\_ios",

          }

connect\_rtr = ConnectHandler(\*\*device1)

output = connect\_rtr.send\_command('sh ip int brief')

with open("int\_brief.txt", "w") as f:

    f.write(output)

connect\_rtr.disconnect()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ python netmiko\_ex3.py

Password:

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ ls

int\_brief.txt  netmiko\_ex1.py  netmiko\_ex2.py  netmiko\_ex3.py  netmiko\_overiew.py

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class1]$ cat int\_brief.txt

Interface              IP-Address      OK? Method Status                Protocol

GigabitEthernet0/0/0   10.220.88.22    YES manual up                    up

GigabitEthernet0/0/1   unassigned      YES NVRAM  administratively down down

Gi0/0/1.146            unassigned      YES manual deleted               down

Gi0/0/1.147            unassigned      YES manual deleted               down

Gi0/0/1.148            unassigned      YES manual deleted               down

Gi0/0/1.149            unassigned      YES TFTP   deleted               down

Gi0/0/1.150            unassigned      YES manual deleted               down

GigabitEthernet0/1/0   unassigned      YES unset  down                  down

GigabitEthernet0/1/1   unassigned      YES unset  down                  down

GigabitEthernet0/1/2   unassigned      YES unset  down                  down

GigabitEthernet0/1/3   unassigned      YES unset  down                  down

Vlan1                  unassigned      YES unset  up                    down

**Pyplus - Class2 - Netmiko Handling Additional Prompts**

vendredi, 22 février 2019

17:14

Let's say we delete this file, ‘testk.txt’ - just some file on my flash. Now notice when I

execute this command on the device, it does not give me back the router prompt. I actually have to hit enter and then it prompts me for additional information, again I have to hit enter. You could call this a multi-line command. There's additional level of prompting, it doesn't all handle it in one operation.

cisco3#delete flash:testk.txt

Delete filename [testk.txt]?

Delete bootflash:/testk.txt? [confirm]

We'll go back to our script and we'll talk about how we can delete this file.

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": password,

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

del\_file = connect\_rtr.send\_command("delete flash:testk.txt", expect\_string=r"[testk.txt]")

del\_file += connect\_rtr.send\_command("", expect\_string=r"confirm")

del\_file += connect\_rtr.send\_command("", expect\_string=r"#")

print(del\_file)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_addpmpt.py

Password:

Delete filename [testk.txt]? Delete bootflash:/testk.txt? [confirm]

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$

The file has got deleted but we do not see what is happening completely because output does not looks good.

By default, Netmiko strips the prompt off. I'm going to do a little trick here to make the output look better. I am going to disable that behavior. Also Netmiko strips the echo of the command off. I'm going to disable that behavior. I'm doing that because I know that in this context, that it actually looks nicer if we actually retrieve the entire output. Let's go ahead and print the output and see what this look like.

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": password,

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

command = "delete flash:/testx.txt"

del\_file = connect\_rtr.send\_command(command, expect\_string=r"[testx.txt]?", strip\_prompt=False, strip\_command=False)

del\_file += connect\_rtr.send\_command("", expect\_string=r"confirm", strip\_prompt=False, strip\_command=False)

del\_file += connect\_rtr.send\_command("", expect\_string=r"#", strip\_prompt=False, strip\_command=False)

print(del\_file)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_addpmpt.py

Password:

delete flash:/testx.txt

Delete filename [testx.txt]?

Delete bootflash:/testx.txt? [confirm]

cisco3#

There is one another way to handle the multi-line command or handling additional prompts. This is instead of using "expect\_string", we use "send\_command\_timing". Here is the code.

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": password,

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

command = "delete flash:/testTofu.txt"

del\_file = connect\_rtr.send\_command\_timing(command, strip\_prompt=False, strip\_command=False)

del\_file += connect\_rtr.send\_command\_timing("", strip\_prompt=False, strip\_command=False)

del\_file += connect\_rtr.send\_command\_timing("", strip\_prompt=False, strip\_command=False)

print(del\_file)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_addpmpt.py

Password:

delete flash:/testTofu.txt

Delete filename [testTofu.txt]?

Delete bootflash:/testTofu.txt? [confirm]

cisco3#

**Pyplus - Class2 - Netmiko delay factor**

mercredi, 6 mars 2019

18:42

Let's go ahead and start discussing global delay factor and delay factor.

The problem we are solving with the global\_delay\_factor or delay\_factor is this problem in screen scraping. It means in general, that we don't have a good determination of when a command is completed when sent over netmiko. This causes all sorts of problems in screen scraping, and you need a way to work around these difficulties. In Netmiko, two really fundamental ways, and they're related, are global delay factor and delay factor.

‘global\_delay\_factor’ you would actually add in as an argument in ConnectHandler. Global delay factor is actually a multiplier of the delays built into Netmiko. In general, and there's a few exceptions to this but they're pretty minor, whatever the delays you had in Netmiko, if you set it to two, you've just doubled them. If you set it to four, you have just quadrupled them. If you set it to 10, you've just multiplied them by a factor of 10. I probably, in general, wouldn't go much beyond 10 on the global delay factor. There are probably other ways that you'd want to solve it if

you had to go beyond multiplying by a factor of 10.

This is going to slow Netmiko down, it should operate the same way. The situations you might

want to think about doing this is, if you try to connect to the device and it sometimes works and

sometimes doesn't work, and it gives you strange messages about "unable to find the trailing

router prompt or the trailing output pattern". These are generally indicators that Netmiko didn't complete what it was doing successfully. A lot of times just slowing Netmiko down, allocating more delay time, more sleep time, will have the problem fixed.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ cat netmiko\_delay.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

"global\_delay\_factor": 4

}

connect\_rtr = ConnectHandler(\*\*device1)

output = connect\_rtr.send\_command("show ip int brief")

print(output)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_delay.py

Password:

Interface IP-Address OK? Method Status Protocol

GigabitEthernet0/0/0 10.220.88.22 YES manual up up

GigabitEthernet0/0/1 unassigned YES NVRAM administratively down down

Gi0/0/1.146 unassigned YES manual deleted down

Gi0/0/1.147 unassigned YES manual deleted down

Gi0/0/1.148 unassigned YES manual deleted down

Gi0/0/1.149 unassigned YES TFTP deleted down

Gi0/0/1.150 unassigned YES manual deleted down

GigabitEthernet0/1/0 unassigned YES unset down down

GigabitEthernet0/1/1 unassigned YES unset down down

GigabitEthernet0/1/2 unassigned YES unset down down

GigabitEthernet0/1/3 unassigned YES unset down down

Vlan1 unassigned YES unset up down

After entering the password, it took sometime to do the netmiko operation globally. Meaning even the login process took sometime.

In addition to global delay factor, there's a separate argument for ‘send\_command’ and ‘send\_command\_timing’, just called ‘delay\_factor’. ‘delay\_factor’ also slows things down, but it only slows it down for the given method. If I had ‘global\_delay\_factor’ commented out and I had ‘delay\_factor=5’ here, it would only slow this one command down. It wouldn't slow down our login process.

It wouldn't slow down our ‘find prompt’ method call. It would just slow down this operation here. Sometimes, for very long-running commands, not only do you need to specify ‘delay\_factor’. You also need to specify ‘max\_loops’. This is how many times Netmiko is going to go through. It's basically got a loop built into it. The loop has sleep built into it. It's going to basically control how many times Netmiko goes through its loop trying to complete the command. That was one of the things you would probably want to do if you start getting ‘delay\_factor’ much over a value of about 10.

Well, what if I have both ‘global\_delay\_factor’ and ‘delay\_factor’, and the same are both active?"

By default, Netmiko's going to pick the larger of them. ‘delay\_factor’ and ‘global\_delay\_factor’ are both geared, in general, towards going slower, not faster. You're actually going to have difficulty making the ‘delay\_factor’ go faster. If you tried to set this to be something less than one, you're probably going to have troubles getting Netmiko to go faster because both ‘global\_delay\_factor’ and ‘delay\_factor’ are geared towards going slower.

Let us see the code for "delay\_factor".

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ cat netmiko\_delay.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

output = connect\_rtr.send\_command("show ip int brief", delay\_factor = 2)

print(output)

Netmiko has both a ‘global\_delay\_factor’, it basically slows down all the sleeps in the Netmiko code. It has a method specific ‘delay\_factor’ in the case of ‘send\_command’ here that'll slow down only the sleeps associated with that one particular method. If you have them both sent, it's going to pick the largest of the two by default.

Pyplus - Class2 - Netmiko TextFSM

jeudi, 7 mars 2019

18:50

Netmiko is inherently connecting two devices via SSH or, in come cases, Telnet, and serial port.

But it's basically using a screen scraping methodology. When we execute show commands, all

that we get back is a block of text. What we generally want back is not just some big block of

text like the output of ‘show ip int brief’. Instead, we want structured data - we want lists and we

want dictionaries, in general.

You have this problem of how do you convert from this block of text that we get back into

structured data? There are various ways that you could potentially accomplish this. For

example, you could write your own regular expression parsing, you could use something like

NAPALM - which has a set of getters available to you that return structured data.

Another solution is to use something known as TextFSM. TextFSM is basically a regular

expression state machine, it's basically a complex regular expression parser. Now, a set of

people have already created a existing TextFSM templates and there's a way you can just

couple those TextFSM templates directly to Netmiko output. Let's go ahead and look at an

example of doing this.

The first thing we have to do is get the setup correctly. We're going to look up ntc-templates.

We're going to use these templates from ntc. We're going to ‘git clone’ this into our environment, so go here, I do ‘git clone’. I copied it. Ensure I copy to my home directory.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 ~]$ git clone <https://github.com/networktocode/ntc-templates>

Cloning into 'ntc-templates'...

remote: Enumerating objects: 14, done.

remote: Counting objects: 100% (14/14), done.

remote: Compressing objects: 100% (14/14), done.

remote: Total 4960 (delta 1), reused 3 (delta 0), pack-reused 4946

Receiving objects: 100% (4960/4960), 1.18 MiB | 9.47 MiB/s, done.

Resolving deltas: 100% (2741/2741), done

If you do not clone to the home directory, you will get the following when you run the python script.

ValueError:

Valid ntc-templates not found, please install <https://github.com/networktocode/ntc-templates>

and then set the NET\_TEXTFSM environment variable to point to the ./ntc-templates/templates

directory

Let us see the code on where to use the textfsm and run this code. You can the see output is not a text but a list that contains directory.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ cat netmiko\_tfsm.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

output = connect\_rtr.send\_command("show ip int brief", use\_textfsm = True)

print(output)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_tfsm.py

Password:

[{'intf': 'GigabitEthernet0/0/0', 'ipaddr': '10.220.88.22', 'status': 'up', 'proto': 'up'}, {'intf': 'GigabitEthernet0/0/1', 'ipaddr': 'unassigned', 'status': 'administratively down', 'proto': 'down'}, {'intf': 'GigabitEthernet0/1/0', 'ipaddr': 'unassigned', 'status': 'down', 'proto': 'down'}, {'intf': 'GigabitEthernet0/1/1', 'ipaddr': 'unassigned', 'status': 'down', 'proto': 'down'}, {'intf': 'GigabitEthernet0/1/2', 'ipaddr': 'unassigned', 'status': 'down', 'proto': 'down'}, {'intf': 'GigabitEthernet0/1/3', 'ipaddr': 'unassigned', 'status': 'down', 'proto': 'down'}, {'intf': 'Vlan1', 'ipaddr': 'unassigned', 'status': 'up', 'proto': 'down'}]

The list that is displayed in hard to read so let use import pprint from the library.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ cat netmiko\_tfsm.py

from netmiko import ConnectHandler

from getpass import getpass

from pprint import pprint

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

output = connect\_rtr.send\_command("show ip int brief", use\_textfsm = True)

pprint(output)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_tfsm.py

Password:

[{'intf': 'GigabitEthernet0/0/0',

'ipaddr': '10.220.88.22',

'proto': 'up',

'status': 'up'},

{'intf': 'GigabitEthernet0/0/1',

'ipaddr': 'unassigned',

'proto': 'down',

'status': 'administratively down'},

{'intf': 'GigabitEthernet0/1/0',

'ipaddr': 'unassigned',

'proto': 'down',

'status': 'down'},

{'intf': 'GigabitEthernet0/1/1',

'ipaddr': 'unassigned',

'proto': 'down',

'status': 'down'},

{'intf': 'GigabitEthernet0/1/2',

'ipaddr': 'unassigned',

'proto': 'down',

'status': 'down'},

{'intf': 'GigabitEthernet0/1/3',

'ipaddr': 'unassigned',

'proto': 'down',

'status': 'down'},

{'intf': 'Vlan1', 'ipaddr': 'unassigned', 'proto': 'down', 'status': 'up'}]

Notice ‘ntc-templates’ has a templates directory. Inside the templates, there's an index file.

These templates here are actually all the templates available. Notice the templates actually map

to a given platform, a given command. One thing you have to make sure of, is you have to make

sure your command exists in that index file.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 ~]$ cd ntc-templates/

(py3\_venv) [sethuvignesh@ip-172-30-0-144 ntc-templates]$ cd templates/

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ ls

alcatel\_sros\_oam\_mac-ping.template cisco\_ios\_show\_ipv6\_interface\_brief.template

alcatel\_sros\_show\_router\_bgp\_routes\_vpn-ipv4.template cisco\_ios\_show\_isdn\_status.template

alcatel\_sros\_show\_service\_id\_base.template cisco\_ios\_show\_isis\_neighbors.template

arista\_eos\_bash\_df\_-h.template cisco\_ios\_show\_lldp\_neighbors\_detail.template

arista\_eos\_show\_boot-config.template cisco\_ios\_show\_lldp\_neighbors.template

arista\_eos\_show\_clock.template cisco\_ios\_show\_mac-address-table.template

arista\_eos\_show\_environment\_cooling.template cisco\_ios\_show\_platform\_diag.template

arista\_eos\_show\_environment\_temperature.template cisco\_ios\_show\_power\_available.template

arista\_eos\_show\_hostname.template cisco\_ios\_show\_power\_status.template

arista\_eos\_show\_interfaces\_status.template cisco\_ios\_show\_power\_supplies.template

arista\_eos\_show\_interfaces.template cisco\_ios\_show\_processes\_cpu.template

arista\_eos\_show\_interfaces\_transceiver\_detail.template cisco\_ios\_show\_redundancy.template

arista\_eos\_show\_interfaces\_transceiver.template cisco\_ios\_show\_route-map.template

arista\_eos\_show\_inventory.template cisco\_ios\_show\_running-config\_partition\_access-list.template

arista\_eos\_show\_ip\_access-lists.template cisco\_ios\_show\_running-config\_partition\_route-map.template

arista\_eos\_show\_ip\_arp.template cisco\_ios\_show\_snmp\_community.template

arista\_eos\_show\_ip\_bgp\_summary.template cisco\_ios\_show\_spanning-tree.template

arista\_eos\_show\_ip\_bgp.template cisco\_ios\_show\_standby\_brief.template

arista\_eos\_show\_ip\_interface\_brief.template cisco\_ios\_show\_standby.template

arista\_eos\_show\_ip\_ospf\_database.template cisco\_ios\_show\_version.template

arista\_eos\_show\_ip\_ospf\_neighbor.template cisco\_ios\_show\_vlan.template

arista\_eos\_show\_ip\_route.template cisco\_ios\_show\_vrf.template

arista\_eos\_show\_isis\_neighbors.template cisco\_ios\_show\_vtp\_status.template

arista\_eos\_show\_lldp\_neighbors\_detail.template cisco\_nxos\_show\_access-lists.template

arista\_eos\_show\_lldp\_neighbors.template cisco\_nxos\_show\_cdp\_neighbors\_detail.template

arista\_eos\_show\_mac\_address-table.template cisco\_nxos\_show\_cdp\_neighbors.template

arista\_eos\_show\_mac\_security\_interface.template cisco\_nxos\_show\_clock.template

arista\_eos\_show\_mac\_security\_mka\_counters.template cisco\_nxos\_show\_configuration\_session\_summary.template

arista\_eos\_show\_mac\_security\_participants\_detail.template cisco\_nxos\_show\_cts\_interface\_all.template

arista\_eos\_show\_mlag.template cisco\_nxos\_show\_cts\_interface\_brief.template

arista\_eos\_show\_reload\_cause.template cisco\_nxos\_show\_environments.template

arista\_eos\_show\_snmp\_community.template cisco\_nxos\_show\_environment\_temperature.template

arista\_eos\_show\_version.template cisco\_nxos\_show\_feature.template

arista\_eos\_show\_vlan.template cisco\_nxos\_show\_fex\_id.template

aruba\_os\_show\_ip\_interface\_brief.template cisco\_nxos\_show\_fex.template

aruba\_os\_show\_ipv6\_interface\_brief.template cisco\_nxos\_show\_flogi\_database.template

avaya\_ers\_show\_interface\_name.template cisco\_nxos\_show\_hostname.template

avaya\_ers\_show\_logging\_config.template cisco\_nxos\_show\_interface\_brief.template

avaya\_ers\_show\_mac-address-table.template cisco\_nxos\_show\_interface\_status.template

avaya\_ers\_show\_mlt.template cisco\_nxos\_show\_interface.template

avaya\_ers\_show\_sys-info.template cisco\_nxos\_show\_interface\_transceiver\_details.template

avaya\_ers\_show\_vlan.template cisco\_nxos\_show\_inventory.template

avaya\_vsp\_show\_software.template cisco\_nxos\_show\_ip\_arp\_detail.template

brocade\_fastiron\_show\_arp.template cisco\_nxos\_show\_ip\_arp.template

brocade\_fastiron\_show\_interfaces\_brief.template cisco\_nxos\_show\_ip\_bgp\_neighbors.template

brocade\_fastiron\_show\_interfaces.template cisco\_nxos\_show\_ip\_bgp\_summary.template

brocade\_fastiron\_show\_lag\_brief.template cisco\_nxos\_show\_ip\_bgp.template

brocade\_fastiron\_show\_lldp\_neighbors\_detail.template cisco\_nxos\_show\_ip\_community-list.template

brocade\_fastiron\_show\_lldp\_neighbors.template cisco\_nxos\_show\_ip\_dhcp\_relay\_address.template

brocade\_fastiron\_show\_mac-address.template cisco\_nxos\_show\_ip\_ospf\_database.template

brocade\_fastiron\_show\_metro.template cisco\_nxos\_show\_ip\_ospf\_neighbor\_vrf.template

brocade\_fastiron\_show\_monitor.template cisco\_nxos\_show\_ip\_route.template

brocade\_fastiron\_show\_running-config\_vlan.template cisco\_nxos\_show\_ipv6\_interface\_brief.template

brocade\_fastiron\_show\_span.template cisco\_nxos\_show\_lldp\_neighbors\_detail.template

brocade\_fastiron\_show\_topo.template cisco\_nxos\_show\_lldp\_neighbors.template

brocade\_fastiron\_show\_trunk.template cisco\_nxos\_show\_mac\_address-table.template

brocade\_fastiron\_show\_version.template cisco\_nxos\_show\_module.template

brocade\_netiron\_show\_interfaces\_brief.template cisco\_nxos\_show\_port-channel\_summary.template

brocade\_netiron\_show\_interfaces.template cisco\_nxos\_show\_processes\_cpu.template

brocade\_netiron\_show\_lag\_brief.template cisco\_nxos\_show\_version.template

brocade\_netiron\_show\_lldp\_neighbors\_detail.template cisco\_nxos\_show\_vlan.template

brocade\_netiron\_show\_metro.template cisco\_nxos\_show\_vpc.template

brocade\_netiron\_show\_monitor.template cisco\_nxos\_show\_vrf.template

brocade\_netiron\_show\_running-config\_interface\_ve.template cisco\_wlc\_ssh\_show\_ap\_config\_general.template

brocade\_netiron\_show\_running-config\_vlan.template cisco\_wlc\_ssh\_show\_ap\_summary.template

brocade\_netiron\_show\_span.template cisco\_wlc\_ssh\_show\_cdp\_neighbors\_detail.template

brocade\_netiron\_show\_topo.template cisco\_wlc\_ssh\_show\_sysinfo.template

checkpoint\_gaia\_fw\_stat.template cisco\_xr\_admin\_show\_controller\_fabric\_health.template

checkpoint\_gaia\_show\_asset\_all.template cisco\_xr\_admin\_show\_environment\_fan.template

checkpoint\_gaia\_show\_domainname.template cisco\_xr\_admin\_show\_platform.template

checkpoint\_gaia\_show\_interfaces\_all.template cisco\_xr\_admin\_show\_vm.template

checkpoint\_gaia\_show\_ipv6\_route.template cisco\_xr\_show\_bfd\_sessions.template

checkpoint\_gaia\_show\_lom.template cisco\_xr\_show\_bgp\_vrf\_all\_ipv4\_unicast\_summary.template

checkpoint\_gaia\_show\_route.template cisco\_xr\_show\_cdp\_neighbors\_detail.template

checkpoint\_gaia\_show\_version\_all.template cisco\_xr\_show\_configuration\_commit\_list.template

cisco\_asa\_dir.template cisco\_xr\_show\_controller\_fabric\_plane\_all.template

cisco\_asa\_show\_access-list.template cisco\_xr\_show\_controllers\_HundredGigabitEthernet.template

cisco\_asa\_show\_asp\_table\_vpn-context\_detail.template cisco\_xr\_show\_controllers.template

cisco\_asa\_show\_crypto\_ikev1\_sa\_detail.template cisco\_xr\_show\_dhcp\_ipv4\_proxy\_binding.template

cisco\_asa\_show\_crypto\_ipsec\_sa.template cisco\_xr\_show\_interface\_brief.template

cisco\_asa\_show\_failover.template cisco\_xr\_show\_interfaces.template

cisco\_asa\_show\_interface.template cisco\_xr\_show\_ip\_bgp\_summary.template

cisco\_asa\_show\_inventory.template cisco\_xr\_show\_ip\_route.template

cisco\_asa\_show\_name.template cisco\_xr\_show\_isis\_neighbors.template

cisco\_asa\_show\_nat.template cisco\_xr\_show\_lldp\_neighbors.template

cisco\_asa\_show\_object-group\_network.template cisco\_xr\_show\_mpls\_ldp\_neighbor\_brief.template

cisco\_asa\_show\_resource\_usage.template cisco\_xr\_show\_ospf\_neighbor.template

cisco\_asa\_show\_route.template cisco\_xr\_show\_pim\_neighbor.template

cisco\_asa\_show\_running-config\_crypto\_ikev1.template cisco\_xr\_show\_processes\_cpu.template

cisco\_asa\_show\_running-config\_crypto\_map.template cisco\_xr\_show\_redundancy\_summary.template

cisco\_asa\_show\_running-config\_ipsec.template cisco\_xr\_show\_rsvp\_neighbors.template

cisco\_asa\_show\_running-config\_object\_network.template cisco\_xr\_show\_version.template

cisco\_asa\_show\_running-config\_tunnel-group.template dell\_force10\_show\_arp.template

cisco\_asa\_show\_version.template dell\_force10\_show\_version.template

cisco\_asa\_show\_vpn-sessiondb\_detail\_l2l.template dell\_force10\_show\_vlan\_brief.template

cisco\_asa\_show\_xlate.template dell\_force10\_show\_vlan.template

cisco\_ios\_dir.template hp\_comware\_display\_arp.template

cisco\_ios\_show\_access-list.template hp\_comware\_display\_clock.template

cisco\_ios\_show\_aliases.template hp\_comware\_display\_mac-address.template

cisco\_ios\_show\_archive.template hp\_comware\_display\_vlan\_brief.template

cisco\_ios\_show\_authentication\_sessions.template hp\_procurve\_show\_arp.template

cisco\_ios\_show\_capability\_feature\_routing.template hp\_procurve\_show\_system.template

cisco\_ios\_show\_cdp\_neighbors\_detail.template hp\_procurve\_show\_tech\_buffers.template

cisco\_ios\_show\_cdp\_neighbors.template hp\_procurve\_show\_vlans.template

cisco\_ios\_show\_clock.template index

cisco\_ios\_show\_controller\_t1.template juniper\_junos\_show\_arp\_no-resolve.template

cisco\_ios\_show\_dot1x\_all.template juniper\_junos\_show\_chassis\_cluster\_interfaces.template

cisco\_ios\_show\_environment\_power\_all.template juniper\_junos\_show\_chassis\_cluster\_status.template

cisco\_ios\_show\_interfaces\_status.template juniper\_junos\_show\_chassis\_firmware.template

cisco\_ios\_show\_interfaces.template juniper\_junos\_show\_interfaces.template

cisco\_ios\_show\_interface\_transceiver.template juniper\_junos\_show\_isis\_adjacency.template

cisco\_ios\_show\_inventory.template juniper\_junos\_show\_ospf\_neighbor.template

cisco\_ios\_show\_ip\_access-lists.template juniper\_junos\_show\_version.template

cisco\_ios\_show\_ip\_arp.template juniper\_screenos\_get\_route.template

cisco\_ios\_show\_ip\_bgp\_summary.template paloalto\_panos\_show\_counter\_global.template

cisco\_ios\_show\_ip\_bgp.template paloalto\_panos\_show\_high-availability\_all.template

cisco\_ios\_show\_ip\_device\_tracking\_all.template paloalto\_panos\_show\_interface\_hardware.template

cisco\_ios\_show\_ip\_eigrp\_neighbors.template paloalto\_panos\_show\_interface\_logical.template

cisco\_ios\_show\_ip\_eigrp\_topology.template paloalto\_panos\_show\_jobs\_all.template

cisco\_ios\_show\_ip\_interface\_brief.template paloalto\_panos\_show\_running\_nat-policy.template

cisco\_ios\_show\_ip\_interface.template paloalto\_panos\_show\_running\_security-policy.template

cisco\_ios\_show\_ip\_mroute.template paloalto\_panos\_show\_system\_info.template

cisco\_ios\_show\_ip\_ospf\_database.template ubiquiti\_edgeswitch\_show\_arp.template

cisco\_ios\_show\_ip\_ospf\_interface\_brief.template ubiquiti\_edgeswitch\_show\_vlan.template

cisco\_ios\_show\_ip\_ospf\_neighbor.template vmware\_nsxv\_show\_ip\_bgp\_neighbors.template

cisco\_ios\_show\_ip\_prefix-list.template vmware\_nsxv\_show\_ip\_route.template

cisco\_ios\_show\_ip\_route.template vyatta\_vyos\_show\_arp.template

cisco\_ios\_show\_ip\_source\_binding.template vyatta\_vyos\_show\_interfaces.template

Pyplus - Class2 - Netmiko Config Change

vendredi, 8 mars 2019

10:05

let's actually make config change, we'll actually change ‘logging buffered 8188' and the method we're going to use is ‘send\_config\_set’. We pass in our config change, go ahead and save that. We're going to just make a single line change

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ cat netmiko\_cfg\_change.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

cfg\_change = 'logging buffered 8188'

output = connect\_rtr.send\_config\_set(cfg\_change)

print(output)

connect\_rtr.disconnect()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_cfg\_change.py

Password:

cisco3#

config term

Enter configuration commands, one per line. End with CNTL/Z.

cisco3(config)#logging buffered 8188

cisco3(config)#end

cisco3#

What if we had a set of changes that we wanted to make . Let us see do the following changes,

Change logging, change time zone, change the logging buffer level to 9000.

In this case instead of having the config as string, we make it as list.

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

**cfg\_change = ['logging buffered 9000','no logging console','clock timezone CET +1']**

output = connect\_rtr.send\_config\_set(cfg\_change)

print(output)

connect\_rtr.disconnect()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_cfg\_change.py

Password:

cisco3#

config term

Enter configuration commands, one per line. End with CNTL/Z.

cisco3(config)#logging buffered 9000

cisco3(config)#no logging console

cisco3(config)#clock timezone CET +1

cisco3(config)#end

cisco3#

you can also configure your changes from a file. Let's go ahead and put this configs line by line in a file. Let say I have the file name as my\_cfg\_changes.txt.

**logging buffered 1000**

**no logging console**

**clock timezone EST -5**

And we send this file using Netmiko command ‘send\_config\_from\_file’ and pass on the value to the argument ‘config\_file=’ .

Let us see the code

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ cat my\_cfg\_changes.txt

logging buffered 10000

no logging console

clock timezone EST -5

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ cat netmiko\_cfg\_change.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

output = connect\_rtr.send\_config\_from\_file(config\_file = 'my\_cfg\_changes.txt')

print(output)

connect\_rtr.disconnect()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_cfg\_change.py

Password:

cisco3#

config term

Enter configuration commands, one per line. End with CNTL/Z.

cisco3(config)#logging buffered 10000

cisco3(config)#no logging console

cisco3(config)#clock timezone EST -5

cisco3(config)#end

cisco3#

The issues you could run into is if you try to send a non config command in a config change.

If you try to do something like this, once again it's just going to be an invalid command. You

could, in Cisco IOS do the following, although I'd probably advise against that because then

you're mixing in Netmiko context where it expects config commands with a show operation.

Now, Netmiko will automatically enter and exit config mode, so you do not need to put either ‘conf t’ nor do you need to put ‘exit’ or ‘end’ inside your set of configuration commands.

You can also potentially run into issues if you're doing a very large set of commands, where Netmiko starts to get really slow. A bit of improvements to be out in Netmiko 2.3.x. Also there's Netmiko's ‘fast\_cli’ mode so you should be able to improve the performance greatly through the combination of Netmiko 2.3.x and potentially ‘fast\_cli=True’.

Pyplus - Class2 - Save Config

vendredi, 8 mars 2019

12:56

We establish the SSH connection. We send a bunch of configuration changes, we print the output. Notice here, at the very end, I call the ‘connect\_rtr.save\_config’ method. That should perform the operation of essentially accomplishing the equivalent of ‘write mem’. It should be supported on a pretty broad set of platforms in Netmiko.We'll go ahead and execute that. Verify that that operates correctly.

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

cfg\_change = connect\_rtr.send\_config\_from\_file(config\_file = 'my\_cfg\_changes.txt')

print(cfg\_change)

output = connect\_rtr.save\_config()

print(output)

connect\_rtr.disconnect()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_cfg\_change\_save.py

Password:

cisco3#

config term

Enter configuration commands, one per line. End with CNTL/Z.

cisco3(config)#logging buffered 8555

cisco3(config)#no logging console

cisco3(config)#clock timezone CET +1

cisco3(config)#end

cisco3#

Building configuration...

[OK]

Pyplus - Class2 - Netmiko Secure Copy

vendredi, 8 mars 2019

11:33

Netmiko and Secure Copy, Netmiko has the capability to do a secure copy file transfer built into it. Let's look at some example code to accomplish that. So ‘import getpass’, we import Netmiko, we import our standard ConnectHandler, we also import this ‘file\_transfer’ function. Now, Netmiko's secure copy mechanism uses an SSH control channel to do certain operations, and then uses a separate secure copy channel to actually accomplish the transfer. You can think of it as somewhat analogous to FTP, where you have a

control channel and you have a data channel.

The first thing we're going to do is we create our dictionary . Here we're going to transfer the file to an IOS device. We then specify the source file we're going to use, and that

‘csk.txt’ is in the current working directory. The destination file, we specify the direction,

whether it's a ‘get’ or a ‘put’, and we specify the remote ‘file\_system’. In the case the remote file system is going to be ‘bootflash’. Then we just create our standard SSH

connection using ConnectHandler, just like we have previously. Then in order to accomplish the file transfer, we call the ‘file\_transfer’ function, we pass in our SSH connection that we just created, we specify the source file, the destination file, the file system, the direction.

Now, the file transfer capability automatically does certain things for you.

It's automatically going to check whether you have sufficient space available. It's automatically going to check whether the file already exists. You have to tell Netmiko whether it's okay to overwrite the existing file. If ‘csk.text’ already exists on the remote NX-OS device, you have to say whether or not it is okay to overwrite the existing, "over\_writing=True".

The file transfer capability also has a built in MD5 in it. It's automatically going to verify that the destination MD5 matches the source MD5, so that the file transfer happened correctly.

One other aspect of the file transfer function is it's idempotent. Before it actually does the file transfer, it checks whether the file is already correct. In this case, does the destination ‘csk.txt’ exist? If it exists, does the MD5 already match? If the MD5 already matched, then Netmiko knows that it has the correct remote file, [and] no file transfer is necessary. It'll basically just say, “Yes, the file already exists, and there's nothing else that we need to do”.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ cat netmiko\_scp.py

from netmiko import ConnectHandler, file\_transfer

from getpass import getpass

from pprint import pprint

password = getpass()

# Define the device details

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": password,

"device\_type": 'cisco\_ios',

}

# Define the file parameters

src\_file = 'csk.txt'

dst\_file = 'csk.txt'

direction = 'put'

file\_system = 'bootflash:'

# Connect to the Device

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

# Perform file transfer operation

scp\_operation = file\_transfer(

connect\_rtr,

source\_file = src\_file,

dest\_file = dst\_file,

direction = direction,

file\_system = file\_system

)

print(scp\_operation)

connect\_rtr.disconnect()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_scp.py

Password:

cisco3#

{'file\_exists': True, 'file\_transferred': True, 'file\_verified': True}

Let us see what happens when you run the script again, meaning we are trying to send the same file again to the router. When we run it a second time we should get the ‘file\_exist=True’, but

‘file\_transferred’ is now false because it was already in the correct state, so it didn't need to do any file transfer, meaning file exists and the md5 checksum is same.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_scp.py

Password:

cisco3#

{'file\_exists': True, '**file\_transferred': False**, 'file\_verified': True}

Now let us grab a file from the router back to our machine. In this case the direction is 'get' and let us change the destiname file name.

# Define the file parameters

src\_file = 'csk.txt'

dst\_file = 'whistle\_podu.txt'

direction = 'get'

file\_system = 'bootflash:'

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_scp.py

Password:

cisco3#

{'file\_exists': True, '**file\_transferred': True**, 'file\_verified': True}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ ls

**csk.txt** my\_cfg\_changes.txt netmiko\_addpmpt.py netmiko\_cfg\_change.py netmiko\_delay.py netmiko\_scp.py netmiko\_tfsm.py **whistle\_podu.txt**

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ md5sum csk.txt whistle\_podu.txt

a6c8a9e2b46a9d537e2baa5af2f84a96 csk.txt

a6c8a9e2b46a9d537e2baa5af2f84a96 whistle\_podu.txt

What are some “gotchas” with the file transfer process? One gotcha is if you're doing a large transfer, your SSH control channel might expire - it might timeout before the transfer is complete. If you're transferring image files and they're going to take a very long time, make sure your VTY timeout is not going to timeout before the transfer is complete.

Another “gotcha” that you can run into is if you're trying to do a transfer via a bastion host and then that's going to be a bit of a problem, because the secure copy channel will not work if you're bouncing through a bastion host.

What can you do on Cisco IOS, Cisco IOS has an inline transfer mode. You can toggle and

change to this inline transfer mode, and then the inline transfer will all happen entirely inside the one SSH channel, and in that case, you can bounce through a bastion host. Now, one caveat with the inline transfer is inline transfer is going to do this TCL based file transfer, and that's only going to support text based files. You will not be able to transfer a binary file via that inline transfer mode.

The one other thing I want to show you is, the number of platforms that are supported on the file transfer is a bit limited. There aren't as many platforms available on that as there are for the standard Netmiko platform. You can see the list of the platforms that file transfers supported on, and you can see there's on the order of about nine of those.

Pyplus - Class2 - using SSH Keys

vendredi, 8 mars 2019

17:50

On the Cisco device, I had to go through the process of setting up an SSH

key for a given user inside of Cisco IOS, which is actually fairly cumbersome to get it to work

properly. But after that's all working, set up, and configured properly, that SSH key is actually bound to a username ‘testuser’ on my Cisco IOS device. I'm going to use the username,‘testuser.

The procedure to setup the keys are discussed in the blog

<https://ccieescort.wordpress.com/2018/09/19/ansible-using-ssh-keys-for-login/>

My SSH keys were in /home/sethuvignesh/.ssh/id\_rsa.pub. Then I specify the ‘use\_keys’: device type, hostname, the username, on the Cisco side that the SSH key is bound to and then ‘use\_keys’ and point to the ‘key\_file’.

Here is the code

from netmiko import ConnectHandler

from pprint import pprint

device1 = {

"host": 'cisco3.lasthop.io',

"device\_type": 'cisco\_ios',

"username": 'testuser',

"use\_keys": True,

"key\_file": '/home/sethuvignesh/.ssh/id\_rsa'

}

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

connect\_rtr.disconnect()

Now, notice, I just connect to the device and print out the prompt. So go ahead, and do

‘netmiko\_keys.py’ and notice it does not prompt me for the password and hopefully if this works correctly we should see the router prompt comeback. So that's using SSH keys.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_keys.py

cisco3#

Now let us use the SSH keys instead of password to securely copy a file to the router.

from netmiko import ConnectHandler, file\_transfer

from pprint import pprint

# Define the device details

device1 = {

"host": 'cisco3.lasthop.io',

"device\_type": 'cisco\_ios',

"username": 'testuser',

"use\_keys": True,

"key\_file": '/home/sethuvignesh/.ssh/id\_rsa'

}

# Define the file parameters

src\_file = 'csk.txt'

dst\_file = 'whistle\_podu.txt'

direction = 'put'

file\_system = 'bootflash:'

# Connect to the Device

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

# Perform file transfer operation

scp\_operation = file\_transfer(

connect\_rtr,

source\_file = src\_file,

dest\_file = dst\_file,

direction = direction,

file\_system = file\_system

)

print(scp\_operation)

connect\_rtr.disconnect()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ python netmiko\_usekeys\_scp.py

cisco3#

{'file\_exists': True, 'file\_transferred': True, 'file\_verified': True}

Pyplus - Class2 - fast\_cli

vendredi, 8 mars 2019

18:58

Let me run the code used to securely copy a file, and see how long it takes to complete the process.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ date; python netmiko\_usekeys\_scp.py; date

Fri Mar 8 **09:58:00** PST 2019

cisco3#

{'file\_exists': True, 'file\_transferred': True, 'file\_verified': True}

Fri Mar 8 **09:58:10** PST 2019

It took around 10 seconds to copy a very small file to the router.

To speed up the process let use "fast\_cli" mode as True. You can see the operation has completed in 3secs.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ date; python netmiko\_fastcli\_cfg.py; date

Fri Mar 8 **10:02:20** PST 2019

cisco3#

{'file\_exists': True, 'file\_transferred': False, 'file\_verified': True}

Fri Mar 8 **10:02:23** PST 2019

The code for this example is below which show using the fast\_cli.

from netmiko import ConnectHandler, file\_transfer

from pprint import pprint

# Define the device details

device1 = {

"host": 'cisco3.lasthop.io',

"device\_type": 'cisco\_ios',

"username": 'testuser',

"use\_keys": True,

"key\_file": '/home/sethuvignesh/.ssh/id\_rsa',

**"fast\_cli": True**

}

# Define the file parameters

src\_file = 'csk.txt'

dst\_file = 'whistle\_podu.txt'

direction = 'put'

file\_system = 'bootflash:'

# Connect to the Device

connect\_rtr = ConnectHandler(\*\*device1)

print(connect\_rtr.find\_prompt())

# Perform file transfer operation

scp\_operation = file\_transfer(

connect\_rtr,

source\_file = src\_file,

dest\_file = dst\_file,

direction = direction,

file\_system = file\_system

)

print(scp\_operation)

connect\_rtr.disconnect()

Let us try this same for a configuration change and see how long it takes. First I will try this without fast\_cli. You can see it took around 19 secs.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ date; python netmiko\_cfg.py; date

Fri Mar 8 **10:09:53** PST 2019

cisco3#

config term

Enter configuration commands, one per line. End with CNTL/Z.

cisco3(config)#logging buffered 8555

cisco3(config)#no logging console

cisco3(config)#clock timezone CET +1

cisco3(config)#end

cisco3#

Fri Mar 8 **10:10:12** PST 2019

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$

Now let me use with fast\_cli, you can see the config change took only 4 secs.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ date; python netmiko\_fastcli\_cfg.py; date

Fri Mar 8 **10:12:24** PST 2019

cisco3#

config term

Enter configuration commands, one per line. End with CNTL/Z.

cisco3(config)#logging buffered 8555

cisco3(config)#no logging console

cisco3(config)#clock timezone CET +1

cisco3(config)#end

cisco3#

Fri Mar 8 **10:12:28** PST 2019

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$

trade off here is really reliability trade off. Yes you're going to go faster, but

you're very likely going to be doing so in a less reliable way. You're going to have to decide

whether that matters. You also have to remember that if you get to above a certain number of devices, you're going to want to do concurrency. The fact that it takes, perhaps 30 or 50

seconds on a single device, with concurrency, you can do a similar 30 to 50 seconds, but

across a whole bunch of devices. So, across a number of devices in, or in can be a larger

number. You definitely have to trade off this, how much is it worth it to go faster versus the

reliability problems that introduces.

In general, I emphasize reliability and I don't emphasize how fast the code executes.

Pyplus - Class3 - Handling Complex Data Structures

vendredi, 22 mars 2019

16:07

let's talk about complex data structures in Python. When I refer to complex datastructures, I am generally referring to lists that contain other lists and dictionaries, or dictionariesthat contain other lists and dictionaries. These could be nested at several levels.Let's look at some examples of these and how you should deal with them. I'm running a script.All this script is doing is just loading a data structure named ‘arp’. If we ‘print(arp)’, we're goingto see that it's a fairly large data structure. The first thing that I almost always do when I'mdealing with complex data structures is I ask myself, “Is it a list, or is it a dictionary?” I look atthis output that I have and I see that I have, at the highest level, I have a list. I could also just doa ‘type’ on it and it would tell me that it's a list.The next thing I do once I know it's a list is I actually look at the length of the list. If that lengthwould've actually just been a length of one, then I would've simply just grabbed the first entry. Iwould've just done the following.Since it's a multiple element list, we can look at the first entry, but the first entry's just going totell us the nature of what's inside of it. Then we'd have to look at subsequent entries. We look atthe first entry here, and we can see that what we get returned are basically these dictionary datastructures.In your code, what you would frequently do is ‘for arp\_entry’ in this data structure. A lot of timeswhen I'm debugging it, I would actually just look at the first entry like this, and then I wouldactually put a break because I want to simplify the problem.

Looking at only one entry helps me simplify the problem. I see that I have that dictionary.Once I look at this inner dictionary, I have to decide what do I want to do with it. Let’s say forexample, in this case, I wanted to grab the MAC address and the IP address. Let's say we didsomething like the following. We'll print just a little banner. We'll print ‘arp\_entry’, but then wewant the key. Let's say we want the MAC address. Then let's say we want to print‘arp\_entry[‘ip’]’. Then, we want to print another banner. Once again, first I'll just do one of them,and I'll use the break to signify that.

There I have my MAC address and the corresponding IP address. If I repeat this and removethe break, then it will do all of them one after the other. Next, I'll actually add another printstatement, just to print another line to make it clearer where the separation is between theentries.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class2]$ ipython -i data\_structures\_test\_arp.py

Python 3.6.7 (default, Dec 21 2018, 20:31:01)

Type 'copyright', 'credits' or 'license' for more information

IPython 7.2.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: dir ()

Out[1]:

['In',

'Out',

'\_',

'\_\_',

'\_\_\_',

'\_\_builtin\_\_',

'\_\_builtins\_\_',

'\_\_doc\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'\_\_spec\_\_',

'\_dh',

'\_i',

'\_i1',

'\_ih',

'\_ii',

'\_iii',

'\_oh',

'arp',

'exit',

'get\_ipython',

'quit']

In [2]: type(arp)

Out[2]: list

In [3]: len(arp)

Out[3]: 14

In [4]: arp[0]

Out[4]:

{'interface': 'FastEthernet4',

'mac': '00:62:EC:29:70:FE',

'ip': '10.220.88.1',

'age': 2.0}

In [5]: arp[1]

Out[5]:

{'interface': 'FastEthernet4',

'mac': '00:24:C4:E9:48:AE',

'ip': '10.220.88.19',

'age': 25.0}

In [6]: for arp\_entry in arp:

...: print('#' \* 12)

...: print(arp\_entry['mac'])

...: print(arp\_entry['ip'])

...: print('#' \* 12)

...: print()

############

00:62:EC:29:70:FE

10.220.88.1

############

############

00:24:C4:E9:48:AE

10.220.88.19

############

############

C8:9C:1D:EA:0E:B6

10.220.88.20

############

############

1C:6A:7A:AF:57:6C

10.220.88.21

############

############

A0:93:51:41:B7:80

10.220.88.22

############

############

50:2F:A8:B1:69:00

10.220.88.23

############

############

52:54:AB:A8:9A:EA

10.220.88.28

############

############

52:54:AB:BE:5B:7B

10.220.88.29

############

############

52:54:AB:71:E1:19

10.220.88.30

############

############

52:54:AB:FB:AF:12

10.220.88.35

############

############

00:01:00:FF:00:01

10.220.88.37

############

############

00:02:00:FF:00:01

10.220.88.38

############

############

64:64:9B:E8:08:C8

10.220.88.39

############

############

EC:38:73:9E:2F:08

10.220.88.42

############

We have a complex data structure. We're processing it.Let's look at an even more complex one. That first one was from NAPALM. Let's look at thisone, which is from Arista's eAPI. We have this ‘test\_code’. It's going to load this interfaces datastructure. We do a print on ‘interfaces’. We're going to see we have a very large data structurethat comes back to us. The first thing I look at again is what is the outermost data structure?Once again, it's a list, so then I want to do a ‘length’ [len(interface)] on it and see what it returnsback.

It only returns a length of one, so then the very first thing I'm going to do is, I'm just going to popthat off. I know I only want the first element, because that's all we have. We'll do ‘interface[0]’,and then we'll look at ‘output’ and see what ‘output’ is.When I look at this one, I see I have a dictionary. When I have a dictionary, what I want to do islook at the keys. I want to see what keys I have in that dictionary. I can see I actually have threekeys. I have a ‘command’ key, I have a ‘result’ key, and I have an ‘encoding’ key. Then, I wouldprobably look at each one of those keys and see if I can get a sense of what those are.The ‘result’ key sounds the most interesting to me, but let's look at each one. There's the ‘result’key. That's probably what we're looking at. We can see we have a bunch of octet data. Let'slook at the last key, which was ‘encoding’, which probably won't be very interesting. It just tellsus that it originally came back as JSON.Now, we're going to want this ‘result’ key. I'm going to save that into a new data structure. I'llactually rename it the same thing, but we're just saving that new innermost part. I'm peeling itback, step by step. If it's a list, I see how long it is. If it's only a single element, I just grab thatsingle element. If it's multiple elements, then I grab one element and look at it or potentially Iloop over it and just grab the first element and use a break. If it's a dictionary, I look at the keys.I start seeing what the keys are and figure out what I need inside that inner dictionary. Onceagain, I'm peeling it back layer by layer. Once you figure out what you need, then you mightwant to convert that over into a function that you just call and does the processing for you.

Let's look at what this new interface is. Once again, we repeat our process. What do we havehere? We have a dictionary again. We can see we have a type dictionary. What are we going to

do with a dictionary? We're going to look at it's keys. See what keys we have. There's an‘interfaces’ key. There's only a single key, so I know that I'm going to want to just get the singlekey. We have another dictionary inside of that. So we'll go, once again, ‘output =’ that,and we'll look at what this is. Once again, we have another dictionary. We're peeling this backlayer by layer. Now we're actually starting to get to something interesting. We have anotherdictionary, but these dictionary keys are interface names. We can pick one of those. Let's try‘Vlan1’, see if there's anything interesting there. We pick one of those and we start looking at it.Now, we would probably need to construct a for loop, so we'd say ‘for intf in output’.You would print out. Actually, I'd want to do ‘.items’ because I want the key name as well. I'd do‘.items;. This would be ‘intf\_name’, call ‘intf\_status’. I could ‘print(intf\_name)’, ‘print(intf\_status)’.And then, I could do ‘break’. Once again, I only want a single one of them. There we have that.On the inner dictionary, we have to decide what we want.

Let's say we want to just get ‘interfaceStatus’. There's a lot of things we could get from this, butlet's just say we want ‘interfaceStatus’ key. I could go up here and say I just want to get the‘interfaceStatus’ key. I messed up my quotes there. Go to the end. Print that. That one'sdisabled. Once again, we could add some sort of banner to make it be prettier, to separate theentries. We'll just put a space for now. Now, let's get rid of our break so that we can look at all ofthem in one go. Here we have the interface name and we have the interface status.Once again, we're going through this process of converting from this complex data structure to the inner things that we need.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ ipython -i data\_structures\_test\_interfaces.py

Python 3.6.7 (default, Dec 21 2018, 20:31:01)

Type 'copyright', 'credits' or 'license' for more information

IPython 7.2.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: dir()

Out[1]:

['In',

'Out',

'\_',

'\_\_',

'\_\_\_',

'\_\_builtin\_\_',

'\_\_builtins\_\_',

'\_\_doc\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'\_\_spec\_\_',

'\_dh',

'\_i',

'\_i1',

'\_ih',

'\_ii',

'\_iii',

'\_oh',

'exit',

'get\_ipython',

'interfaces',

'quit']

In [8]: output1 = interfaces[0]

In [9]: print(output1)

{'command': 'show interfaces', 'result': {'interfaces': {'Management1': {'lastStatusChangeTimestamp': 1538591522.8171098, 'name': 'Management1', 'interfaceStatus': 'disabled', 'autoNegotiate': 'off', 'burnedInAddress': '52:54:ab:02:a1:10', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 1500, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 1000000000, 'forwardingModel': 'routed', 'lineProtocolStatus': 'down', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 2, 'totalOutErrors': 0, 'inMulticastPkts': 0, 'counterRefreshTime': 1539281777.827566, 'inBroadcastPkts': 0, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 0, 'outDiscards': 0, 'outOctets': 0, 'inUcastPkts': 0, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 0, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:10', 'description': ''}, 'Vlan1': {'lastStatusChangeTimestamp': 1538591527.373837, 'name': 'Vlan1', 'interfaceStatus': 'connected', 'burnedInAddress': '52:54:ab:be:5b:7b', 'mtu': 1500, 'hardware': 'vlan', 'bandwidth': 0, 'forwardingModel': 'routed', 'lineProtocolStatus': 'up', 'interfaceAddress': [{'secondaryIpsOrderedList': [], 'broadcastAddress': '255.255.255.255', 'secondaryIps': {}, 'primaryIp': {'maskLen': 24, 'address': '10.220.88.29'}, 'virtualIp': {'maskLen': 0, 'address': '0.0.0.0'}}], 'physicalAddress': '52:54:ab:be:5b:7b', 'description': ''}, 'Ethernet2': {'lastStatusChangeTimestamp': 1538591527.2428443, 'name': 'Ethernet2', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:12', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498763, 'counterRefreshTime': 1539281777.849249, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78083049, 'outDiscards': 0, 'outOctets': 4027630, 'inUcastPkts': 38214, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23018, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:12', 'description': ''}, 'Ethernet3': {'lastStatusChangeTimestamp': 1538591527.243047, 'name': 'Ethernet3', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:13', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498769, 'counterRefreshTime': 1539281777.875926, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78084075, 'outDiscards': 0, 'outOctets': 4026892, 'inUcastPkts': 38215, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23012, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:13', 'description': ''}, 'Ethernet1': {'lastStatusChangeTimestamp': 1538591527.2426362, 'name': 'Ethernet1', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:11', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 1, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498763, 'counterRefreshTime': 1539281777.867376, 'inBroadcastPkts': 4170, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 76679169, 'outDiscards': 0, 'outOctets': 5431798, 'inUcastPkts': 22895, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 15320, 'outMulticastPkts': 23018, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:11', 'description': ''}, 'Ethernet6': {'lastStatusChangeTimestamp': 1538591527.2436152, 'name': 'Ethernet6', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:16', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498768, 'counterRefreshTime': 1539281777.897336, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78083936, 'outDiscards': 0, 'outOctets': 4026892, 'inUcastPkts': 38215, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23012, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:16', 'description': ''}, 'Ethernet7': {'lastStatusChangeTimestamp': 1538591527.243805, 'name': 'Ethernet7', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:17', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498769, 'counterRefreshTime': 1539281777.837162, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78083771, 'outDiscards': 0, 'outOctets': 4026769, 'inUcastPkts': 38214, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23011, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:17', 'description': ''}, 'Ethernet4': {'lastStatusChangeTimestamp': 1538591527.243236, 'name': 'Ethernet4', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:14', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'lastClear': 1538591421.972857, 'inMulticastPkts': 498767, 'counterRefreshTime': 1539281777.858641, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78083541, 'outDiscards': 0, 'outOctets': 4027138, 'inUcastPkts': 38214, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23014, 'totalInErrors': 0, 'inDiscards': 0, 'totalOutErrors': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:14', 'description': ''}, 'Ethernet5': {'lastStatusChangeTimestamp': 1538591527.2434251, 'name': 'Ethernet5', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:15', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498770, 'counterRefreshTime': 1539281777.884441, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78084198, 'outDiscards': 0, 'outOctets': 4026769, 'inUcastPkts': 38215, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23011, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:15', 'description': ''}}}, 'encoding': 'json'}

In [18]: print(output1.keys())

dict\_keys(['command', 'result', 'encoding'])

In [19]: print(output1['command'])

show interfaces

In [20]: print(output1['result'])

{'interfaces': {'Management1': {'lastStatusChangeTimestamp': 1538591522.8171098, 'name': 'Management1', 'interfaceStatus': 'disabled', 'autoNegotiate': 'off', 'burnedInAddress': '52:54:ab:02:a1:10', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 1500, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 1000000000, 'forwardingModel': 'routed', 'lineProtocolStatus': 'down', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 2, 'totalOutErrors': 0, 'inMulticastPkts': 0, 'counterRefreshTime': 1539281777.827566, 'inBroadcastPkts': 0, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 0, 'outDiscards': 0, 'outOctets': 0, 'inUcastPkts': 0, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 0, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:10', 'description': ''}, 'Vlan1': {'lastStatusChangeTimestamp': 1538591527.373837, 'name': 'Vlan1', 'interfaceStatus': 'connected', 'burnedInAddress': '52:54:ab:be:5b:7b', 'mtu': 1500, 'hardware': 'vlan', 'bandwidth': 0, 'forwardingModel': 'routed', 'lineProtocolStatus': 'up', 'interfaceAddress': [{'secondaryIpsOrderedList': [], 'broadcastAddress': '255.255.255.255', 'secondaryIps': {}, 'primaryIp': {'maskLen': 24, 'address': '10.220.88.29'}, 'virtualIp': {'maskLen': 0, 'address': '0.0.0.0'}}], 'physicalAddress': '52:54:ab:be:5b:7b', 'description': ''}, 'Ethernet2': {'lastStatusChangeTimestamp': 1538591527.2428443, 'name': 'Ethernet2', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:12', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498763, 'counterRefreshTime': 1539281777.849249, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78083049, 'outDiscards': 0, 'outOctets': 4027630, 'inUcastPkts': 38214, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23018, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:12', 'description': ''}, 'Ethernet3': {'lastStatusChangeTimestamp': 1538591527.243047, 'name': 'Ethernet3', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:13', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498769, 'counterRefreshTime': 1539281777.875926, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78084075, 'outDiscards': 0, 'outOctets': 4026892, 'inUcastPkts': 38215, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23012, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:13', 'description': ''}, 'Ethernet1': {'lastStatusChangeTimestamp': 1538591527.2426362, 'name': 'Ethernet1', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:11', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 1, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498763, 'counterRefreshTime': 1539281777.867376, 'inBroadcastPkts': 4170, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 76679169, 'outDiscards': 0, 'outOctets': 5431798, 'inUcastPkts': 22895, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 15320, 'outMulticastPkts': 23018, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:11', 'description': ''}, 'Ethernet6': {'lastStatusChangeTimestamp': 1538591527.2436152, 'name': 'Ethernet6', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:16', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498768, 'counterRefreshTime': 1539281777.897336, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78083936, 'outDiscards': 0, 'outOctets': 4026892, 'inUcastPkts': 38215, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23012, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:16', 'description': ''}, 'Ethernet7': {'lastStatusChangeTimestamp': 1538591527.243805, 'name': 'Ethernet7', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:17', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498769, 'counterRefreshTime': 1539281777.837162, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78083771, 'outDiscards': 0, 'outOctets': 4026769, 'inUcastPkts': 38214, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23011, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:17', 'description': ''}, 'Ethernet4': {'lastStatusChangeTimestamp': 1538591527.243236, 'name': 'Ethernet4', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:14', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'lastClear': 1538591421.972857, 'inMulticastPkts': 498767, 'counterRefreshTime': 1539281777.858641, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78083541, 'outDiscards': 0, 'outOctets': 4027138, 'inUcastPkts': 38214, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23014, 'totalInErrors': 0, 'inDiscards': 0, 'totalOutErrors': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:14', 'description': ''}, 'Ethernet5': {'lastStatusChangeTimestamp': 1538591527.2434251, 'name': 'Ethernet5', 'interfaceStatus': 'connected', 'autoNegotiate': 'unknown', 'burnedInAddress': '52:54:ab:02:a1:15', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 9214, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 0, 'forwardingModel': 'bridged', 'lineProtocolStatus': 'up', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 1, 'totalOutErrors': 0, 'inMulticastPkts': 498770, 'counterRefreshTime': 1539281777.884441, 'inBroadcastPkts': 4171, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 78084198, 'outDiscards': 0, 'outOctets': 4026769, 'inUcastPkts': 38215, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 23011, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:15', 'description': ''}}}

In [21]: print(output1['encoding'])

In [24]: output2 = output1['result']

In [25]: output2

Out[25]:

{'interfaces': {'Management1': {'lastStatusChangeTimestamp': 1538591522.8171098,

'name': 'Management1',

'interfaceStatus': 'disabled',

'autoNegotiate': 'off',

'burnedInAddress': '52:54:ab:02:a1:10',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 1500,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 1000000000,

'forwardingModel': 'routed',

'lineProtocolStatus': 'down',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 2,

'totalOutErrors': 0,

'inMulticastPkts': 0,

'counterRefreshTime': 1539281777.827566,

'inBroadcastPkts': 0,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 0,

'outDiscards': 0,

'outOctets': 0,

'inUcastPkts': 0,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 0,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:10',

'description': ''},

'Vlan1': {'lastStatusChangeTimestamp': 1538591527.373837,

'name': 'Vlan1',

'interfaceStatus': 'connected',

'burnedInAddress': '52:54:ab:be:5b:7b',

'mtu': 1500,

'hardware': 'vlan',

'bandwidth': 0,

'forwardingModel': 'routed',

'lineProtocolStatus': 'up',

'interfaceAddress': [{'secondaryIpsOrderedList': [],

'broadcastAddress': '255.255.255.255',

'secondaryIps': {},

'primaryIp': {'maskLen': 24, 'address': '10.220.88.29'},

'virtualIp': {'maskLen': 0, 'address': '0.0.0.0'}}],

'physicalAddress': '52:54:ab:be:5b:7b',

'description': ''},

'Ethernet2': {'lastStatusChangeTimestamp': 1538591527.2428443,

'name': 'Ethernet2',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:12',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498763,

'counterRefreshTime': 1539281777.849249,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78083049,

'outDiscards': 0,

'outOctets': 4027630,

'inUcastPkts': 38214,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23018,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:12',

'description': ''},

'Ethernet3': {'lastStatusChangeTimestamp': 1538591527.243047,

'name': 'Ethernet3',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:13',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498769,

'counterRefreshTime': 1539281777.875926,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78084075,

'outDiscards': 0,

'outOctets': 4026892,

'inUcastPkts': 38215,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23012,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:13',

'description': ''},

'Ethernet1': {'lastStatusChangeTimestamp': 1538591527.2426362,

'name': 'Ethernet1',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:11',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 1,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498763,

'counterRefreshTime': 1539281777.867376,

'inBroadcastPkts': 4170,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 76679169,

'outDiscards': 0,

'outOctets': 5431798,

'inUcastPkts': 22895,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 15320,

'outMulticastPkts': 23018,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:11',

'description': ''},

'Ethernet6': {'lastStatusChangeTimestamp': 1538591527.2436152,

'name': 'Ethernet6',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:16',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498768,

'counterRefreshTime': 1539281777.897336,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78083936,

'outDiscards': 0,

'outOctets': 4026892,

'inUcastPkts': 38215,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23012,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:16',

'description': ''},

'Ethernet7': {'lastStatusChangeTimestamp': 1538591527.243805,

'name': 'Ethernet7',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:17',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498769,

'counterRefreshTime': 1539281777.837162,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78083771,

'outDiscards': 0,

'outOctets': 4026769,

'inUcastPkts': 38214,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23011,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:17',

'description': ''},

'Ethernet4': {'lastStatusChangeTimestamp': 1538591527.243236,

'name': 'Ethernet4',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:14',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'lastClear': 1538591421.972857,

'inMulticastPkts': 498767,

'counterRefreshTime': 1539281777.858641,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78083541,

'outDiscards': 0,

'outOctets': 4027138,

'inUcastPkts': 38214,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23014,

'totalInErrors': 0,

'inDiscards': 0,

'totalOutErrors': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:14',

'description': ''},

'Ethernet5': {'lastStatusChangeTimestamp': 1538591527.2434251,

'name': 'Ethernet5',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:15',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498770,

'counterRefreshTime': 1539281777.884441,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78084198,

'outDiscards': 0,

'outOctets': 4026769,

'inUcastPkts': 38215,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23011,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:15',

'description': ''}}}

In [26]: type(output2)

Out[26]: dict

In [27]: output2.keys()

Out[27]: dict\_keys(['interfaces'])

In [28]: output2['interfaces']

Out[28]:

{'Management1': {'lastStatusChangeTimestamp': 1538591522.8171098,

'name': 'Management1',

'interfaceStatus': 'disabled',

'autoNegotiate': 'off',

'burnedInAddress': '52:54:ab:02:a1:10',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 1500,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 1000000000,

'forwardingModel': 'routed',

'lineProtocolStatus': 'down',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 2,

'totalOutErrors': 0,

'inMulticastPkts': 0,

'counterRefreshTime': 1539281777.827566,

'inBroadcastPkts': 0,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 0,

'outDiscards': 0,

'outOctets': 0,

'inUcastPkts': 0,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 0,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:10',

'description': ''},

'Vlan1': {'lastStatusChangeTimestamp': 1538591527.373837,

'name': 'Vlan1',

'interfaceStatus': 'connected',

'burnedInAddress': '52:54:ab:be:5b:7b',

'mtu': 1500,

'hardware': 'vlan',

'bandwidth': 0,

'forwardingModel': 'routed',

'lineProtocolStatus': 'up',

'interfaceAddress': [{'secondaryIpsOrderedList': [],

'broadcastAddress': '255.255.255.255',

'secondaryIps': {},

'primaryIp': {'maskLen': 24, 'address': '10.220.88.29'},

'virtualIp': {'maskLen': 0, 'address': '0.0.0.0'}}],

'physicalAddress': '52:54:ab:be:5b:7b',

'description': ''},

'Ethernet2': {'lastStatusChangeTimestamp': 1538591527.2428443,

'name': 'Ethernet2',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:12',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498763,

'counterRefreshTime': 1539281777.849249,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78083049,

'outDiscards': 0,

'outOctets': 4027630,

'inUcastPkts': 38214,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23018,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:12',

'description': ''},

'Ethernet3': {'lastStatusChangeTimestamp': 1538591527.243047,

'name': 'Ethernet3',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:13',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498769,

'counterRefreshTime': 1539281777.875926,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78084075,

'outDiscards': 0,

'outOctets': 4026892,

'inUcastPkts': 38215,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23012,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:13',

'description': ''},

'Ethernet1': {'lastStatusChangeTimestamp': 1538591527.2426362,

'name': 'Ethernet1',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:11',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 1,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498763,

'counterRefreshTime': 1539281777.867376,

'inBroadcastPkts': 4170,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 76679169,

'outDiscards': 0,

'outOctets': 5431798,

'inUcastPkts': 22895,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 15320,

'outMulticastPkts': 23018,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:11',

'description': ''},

'Ethernet6': {'lastStatusChangeTimestamp': 1538591527.2436152,

'name': 'Ethernet6',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:16',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498768,

'counterRefreshTime': 1539281777.897336,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78083936,

'outDiscards': 0,

'outOctets': 4026892,

'inUcastPkts': 38215,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23012,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:16',

'description': ''},

'Ethernet7': {'lastStatusChangeTimestamp': 1538591527.243805,

'name': 'Ethernet7',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:17',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498769,

'counterRefreshTime': 1539281777.837162,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78083771,

'outDiscards': 0,

'outOctets': 4026769,

'inUcastPkts': 38214,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23011,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:17',

'description': ''},

'Ethernet4': {'lastStatusChangeTimestamp': 1538591527.243236,

'name': 'Ethernet4',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:14',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'lastClear': 1538591421.972857,

'inMulticastPkts': 498767,

'counterRefreshTime': 1539281777.858641,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78083541,

'outDiscards': 0,

'outOctets': 4027138,

'inUcastPkts': 38214,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23014,

'totalInErrors': 0,

'inDiscards': 0,

'totalOutErrors': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:14',

'description': ''},

'Ethernet5': {'lastStatusChangeTimestamp': 1538591527.2434251,

'name': 'Ethernet5',

'interfaceStatus': 'connected',

'autoNegotiate': 'unknown',

'burnedInAddress': '52:54:ab:02:a1:15',

'loopbackMode': 'loopbackNone',

'interfaceStatistics': {'inBitsRate': 0.0,

'inPktsRate': 0.0,

'outBitsRate': 0.0,

'updateInterval': 300.0,

'outPktsRate': 0.0},

'mtu': 9214,

'hardware': 'ethernet',

'duplex': 'duplexFull',

'bandwidth': 0,

'forwardingModel': 'bridged',

'lineProtocolStatus': 'up',

'interfaceCounters': {'outBroadcastPkts': 0,

'linkStatusChanges': 1,

'totalOutErrors': 0,

'inMulticastPkts': 498770,

'counterRefreshTime': 1539281777.884441,

'inBroadcastPkts': 4171,

'outputErrorsDetail': {'deferredTransmissions': 0,

'txPause': 0,

'collisions': 0,

'lateCollisions': 0},

'inOctets': 78084198,

'outDiscards': 0,

'outOctets': 4026769,

'inUcastPkts': 38215,

'inputErrorsDetail': {'runtFrames': 0,

'rxPause': 0,

'fcsErrors': 0,

'alignmentErrors': 0,

'giantFrames': 0,

'symbolErrors': 0},

'outUcastPkts': 0,

'outMulticastPkts': 23011,

'totalInErrors': 0,

'inDiscards': 0},

'interfaceAddress': [],

'physicalAddress': '52:54:ab:02:a1:15',

'description': ''}}

In [29]: output3 = output2['interfaces']

In [30]: type(output3)

Out[30]: dict

In [32]: output3.keys()

Out[32]: dict\_keys(['Management1', 'Vlan1', 'Ethernet2', 'Ethernet3', 'Ethernet1', 'Ethernet6', 'Ethernet7', 'Ethernet4', 'Ethernet5'])

In [36]: output3['Vlan1']

Out[36]:

{'lastStatusChangeTimestamp': 1538591527.373837,

'name': 'Vlan1',

'interfaceStatus': 'connected',

'burnedInAddress': '52:54:ab:be:5b:7b',

'mtu': 1500,

'hardware': 'vlan',

'bandwidth': 0,

'forwardingModel': 'routed',

'lineProtocolStatus': 'up',

'interfaceAddress': [{'secondaryIpsOrderedList': [],

'broadcastAddress': '255.255.255.255',

'secondaryIps': {},

'primaryIp': {'maskLen': 24, 'address': '10.220.88.29'},

'virtualIp': {'maskLen': 0, 'address': '0.0.0.0'}}],

'physicalAddress': '52:54:ab:be:5b:7b',

'description': ''}

In [50]: for intf\_name, intf\_status in output3.items():

...: print(intf\_name)

...: print(intf\_status)

...: break

...:

Management1

{'lastStatusChangeTimestamp': 1538591522.8171098, 'name': 'Management1', 'interfaceStatus': 'disabled', 'autoNegotiate': 'off', 'burnedInAddress': '52:54:ab:02:a1:10', 'loopbackMode': 'loopbackNone', 'interfaceStatistics': {'inBitsRate': 0.0, 'inPktsRate': 0.0, 'outBitsRate': 0.0, 'updateInterval': 300.0, 'outPktsRate': 0.0}, 'mtu': 1500, 'hardware': 'ethernet', 'duplex': 'duplexFull', 'bandwidth': 1000000000, 'forwardingModel': 'routed', 'lineProtocolStatus': 'down', 'interfaceCounters': {'outBroadcastPkts': 0, 'linkStatusChanges': 2, 'totalOutErrors': 0, 'inMulticastPkts': 0, 'counterRefreshTime': 1539281777.827566, 'inBroadcastPkts': 0, 'outputErrorsDetail': {'deferredTransmissions': 0, 'txPause': 0, 'collisions': 0, 'lateCollisions': 0}, 'inOctets': 0, 'outDiscards': 0, 'outOctets': 0, 'inUcastPkts': 0, 'inputErrorsDetail': {'runtFrames': 0, 'rxPause': 0, 'fcsErrors': 0, 'alignmentErrors': 0, 'giantFrames': 0, 'symbolErrors': 0}, 'outUcastPkts': 0, 'outMulticastPkts': 0, 'totalInErrors': 0, 'inDiscards': 0}, 'interfaceAddress': [], 'physicalAddress': '52:54:ab:02:a1:10', 'description': ''}

In [51]: for intf\_name, intf\_status in output3.items():

...: print(intf\_name)

...: print(intf\_status['interfaceStatus'])

...: break

...:

...:

Management1

disabled

In [53]: for intf\_name, intf\_status in output3.items():

...: print('#' \* 12)

...: print(intf\_name)

...: print(intf\_status['interfaceStatus'])

...: print('#' \* 12)

...: print()

...:

...:

...:

############

Management1

disabled

############

############

Vlan1

connected

############

############

Ethernet2

connected

############

############

Ethernet3

connected

############

############

Ethernet1

connected

############

############

Ethernet6

connected

############

############

Ethernet7

connected

############

############

Ethernet4

connected

############

############

Ethernet5

connected

############

We'll take this, what welearned from that, and then let's go into this program where I actually loaded this data from and see if we can reconstruct that into something that we'd ultimately make be a function.

### Load data collected into the python file first and then add the following script to bottom of that file###

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat data\_structure\_handling.py

### data collected was added but not shown in the script for easy reading###

output1 = interfaces[0]

output2 = output1['result']

output3 = output2['interfaces']

for intf\_name, intf\_status in output3.items():

print('#' \* 12)

print(intf\_name)

print(intf\_status['interfaceStatus'])

print('#' \* 12)

print()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python data\_structure\_handling.py

############

Management1

disabled

############

############

Vlan1

connected

############

############

Ethernet2

connected

############

############

Ethernet3

connected

############

############

Ethernet1

connected

############

############

Ethernet6

connected

############

############

Ethernet7

connected

############

############

Ethernet4

connected

############

############

Ethernet5

connected

############

Pyplus - Class3 - Changing Data Structure Format

vendredi, 22 mars 2019

21:14

you frequently have to do with complex data structures is you frequently need toconvert them from being one format to being a different format.Let's look at a little example of that. We'll execute this ‘test\_lldp’. We'll do a ‘dir’. We'll see wehave this ‘lldp’ variable. This came from NAPALM. We can see our outermost data structure is adictionary. What are we going to look at? We're going to look at the keys. We'll pick one of thoseinterfaces, and we'll look at that interface. Pick ‘Eth2/1’, here's the LLDP entry from thatinterface. Notice this returns a list. Our outermost data structure's a dictionary where we havethe interface names are these keys. We have this inner dictionary.Say we wanted to convert that so that we didn't have the keys as dictionary names, but we hada single list where each of the elements in the list was a dictionary. We're going to convert it to anew data structure that looks something like the following where we have list, dictionary,dictionary, dictionary, until we have covered all our entries.In the inner dictionary, let's just say in our inner dictionary, we're going to have the interfacename, which will be one of these. Then we'll grab just the remote port.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ ipython -i data\_structures\_test\_lldp.py

Python 3.6.7 (default, Dec 21 2018, 20:31:01)

Type 'copyright', 'credits' or 'license' for more information

IPython 7.2.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: dir()

Out[1]:

['In',

'Out',

'\_',

'\_\_',

'\_\_\_',

'\_\_builtin\_\_',

'\_\_builtins\_\_',

'\_\_doc\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'\_\_spec\_\_',

'\_dh',

'\_i',

'\_i1',

'\_ih',

'\_ii',

'\_iii',

'\_oh',

'exit',

'get\_ipython',

'lldp',

'quit']

In [2]: type(lldp)

Out[2]: dict

In [3]: len(lldp)

Out[3]: 4

In [4]: lldp.keys()

Out[4]: dict\_keys(['Eth2/1', 'Eth2/2', 'Eth2/3', 'Eth2/4'])

In [5]: lldp['Eth2/1']

Out[5]:

[{'remote\_chassis\_id': '2C:C2:60:54:DC:2C',

'parent\_interface': '',

'remote\_port': 'Ethernet2/1',

'remote\_port\_description': 'Ethernet2/1',

'remote\_system\_name': 'nxos2.twb-tech.com',

'remote\_system\_description': 'Cisco NX-OS(tm) titanium, Software (titanium-d1), Version 7.3(1)D1(1), Interim version 7.3(1)D1(0.10), RELEASE SOFTWARE Copyright (c) 2002-2013, 2015 by Cisco Systems, Inc. Compiled 1/11/2016 16:00:00',

'remote\_system\_capab': 'B, R',

'remote\_system\_enable\_capab': 'B, R'}]

In [6]: lldp.items()

Out[6]: dict\_items([('Eth2/1', [{'remote\_chassis\_id': '2C:C2:60:54:DC:2C', 'parent\_interface': '', 'remote\_port': 'Ethernet2/1', 'remote\_port\_description': 'Ethernet2/1', 'remote\_system\_name': 'nxos2.twb-tech.com', 'remote\_system\_description': 'Cisco NX-OS(tm) titanium, Software (titanium-d1), Version 7.3(1)D1(1), Interim version 7.3(1)D1(0.10), RELEASE SOFTWARE Copyright (c) 2002-2013, 2015 by Cisco Systems, Inc. Compiled 1/11/2016 16:00:00', 'remote\_system\_capab': 'B, R', 'remote\_system\_enable\_capab': 'B, R'}]), ('Eth2/2', [{'remote\_chassis\_id': '2C:C2:60:64:E1:5F', 'parent\_interface': '', 'remote\_port': 'Ethernet2/2', 'remote\_port\_description': 'Ethernet2/2', 'remote\_system\_name': 'nxos2.twb-tech.com', 'remote\_system\_description': 'Cisco NX-OS(tm) titanium, Software (titanium-d1), Version 7.3(1)D1(1), Interim version 7.3(1)D1(0.10), RELEASE SOFTWARE Copyright (c) 2002-2013, 2015 by Cisco Systems, Inc. Compiled 1/11/2016 16:00:00', 'remote\_system\_capab': 'B, R', 'remote\_system\_enable\_capab': 'B, R'}]), ('Eth2/3', [{'remote\_chassis\_id': '2C:C2:60:72:61:7B', 'parent\_interface': '', 'remote\_port': 'Ethernet2/3', 'remote\_port\_description': 'Ethernet2/3', 'remote\_system\_name': 'nxos2.twb-tech.com', 'remote\_system\_description': 'Cisco NX-OS(tm) titanium, Software (titanium-d1), Version 7.3(1)D1(1), Interim version 7.3(1)D1(0.10), RELEASE SOFTWARE Copyright (c) 2002-2013, 2015 by Cisco Systems, Inc. Compiled 1/11/2016 16:00:00', 'remote\_system\_capab': 'B, R', 'remote\_system\_enable\_capab': 'B, R'}]), ('Eth2/4', [{'remote\_chassis\_id': '2C:C2:60:70:69:DA', 'parent\_interface': '', 'remote\_port': 'Ethernet2/4', 'remote\_port\_description': 'Ethernet2/4', 'remote\_system\_name': 'nxos2.twb-tech.com', 'remote\_system\_description': 'Cisco NX-OS(tm) titanium, Software (titanium-d1), Version 7.3(1)D1(1), Interim version 7.3(1)D1(0.10), RELEASE SOFTWARE Copyright (c) 2002-2013, 2015 by Cisco Systems, Inc. Compiled 1/11/2016 16:00:00', 'remote\_system\_capab': 'B, R', 'remote\_system\_enable\_capab': 'B, R'}])])

Now I'm going to actually copy this and take it back to my code because at this point, it's going to get too hard for me towork in this - in the interpreter shell. I need to work on it in a program.We know we have the keys. The first thing we're going to do is we're going to loop over these.We're going to do, for key value, an lldp.items.

We're going to print the key,which should be the interface name, and we're going to print the value. Once again, to simplifythe problem, I'm only going to look at one element. Always be thinking about how you cansimplify the problem. We'll see if that executes properly. It does. Here's our interface name, andhere we have a list of entries. I'm also going to import pretty print so we can print this nicer. Remember what we're doing here.We're trying to convert this data structure from one form to another form. We're going to goahead and do pretty print on value 0. Go ahead and execute that. There we see the output.Here's our interface name. Here's our corresponding LLDP entry for that.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python data\_structures\_test\_lldp.py

### data collected are not shown for easy reading###

from pprint import pprint

for intf\_name, remote\_system in lldp.items():

pprint(intf\_name)

pprint(remote\_system)

break

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python data\_structures\_data\_format\_change.py

'Eth2/1'

[{'parent\_interface': '',

'remote\_chassis\_id': '2C:C2:60:54:DC:2C',

'remote\_port': 'Ethernet2/1',

'remote\_port\_description': 'Ethernet2/1',

'remote\_system\_capab': 'B, R',

'remote\_system\_description': 'Cisco NX-OS(tm) titanium, Software '

'(titanium-d1), Version 7.3(1)D1(1), Interim '

'version 7.3(1)D1(0.10), RELEASE SOFTWARE '

'Copyright (c) 2002-2013, 2015 by Cisco '

'Systems, Inc. Compiled 1/11/2016 16:00:00',

'remote\_system\_enable\_capab': 'B, R',

'remote\_system\_name': 'nxos2.twb-tech.com'}]

We go back into our program, and we continue on this process, where once again, we're tryingto convert this data structure. We'll say new list equals a blank list [new\_list = []]. Then we'regoing to make a ‘new\_entry’. This is going to be our internal dictionary that we're going to create.The first field is going to be a key known as ‘intf\_name’. The interface name will actually be thatkey value. I'm going to rename it to make my code more readable. We'll call this interface name‘intf\_name’. Then we're going to want the second thing to be the ‘remote\_port’. The LLDP entryfrom the remote device, what port on the remote device are we connected to. Once again, I'm going to do my break, because I only want to look at a single example. I want to simplify the problem. Go ahead and execute that

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat data\_structures\_data\_format\_change.py

### data collected are not shown for easy reading###

from pprint import pprint

new\_list = []

for intf\_name, lldp\_info in lldp.items():

remote\_system = lldp\_info[0]

new\_entry = {

"local\_intf": intf\_name,

"remote\_system": remote\_system['remote\_system\_name'] ,

}

new\_list.append(new\_entry)

break

pprint(new\_list)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python data\_structures\_data\_format\_change.py

[{'local\_intf': 'Eth2/1', 'remote\_system': 'nxos2.twb-tech.com'}]

Now let's remove our break point and see if we have it for all the entries.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat data\_structures\_data\_format\_change.py

### data collected are not shown for easy reading###

from pprint import pprint

new\_list = []

for intf\_name, lldp\_info in lldp.items():

remote\_system = lldp\_info[0]

new\_entry = {

"local\_intf": intf\_name,

"remote\_system": remote\_system['remote\_system\_name'] ,

}

new\_list.append(new\_entry)

pprint(new\_list)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python data\_structures\_data\_format\_change.py

[{'local\_intf': 'Eth2/1', 'remote\_system': 'nxos2.twb-tech.com'},

{'local\_intf': 'Eth2/2', 'remote\_system': 'nxos2.twb-tech.com'},

{'local\_intf': 'Eth2/3', 'remote\_system': 'nxos2.twb-tech.com'},

{'local\_intf': 'Eth2/4', 'remote\_system': 'nxos2.twb-tech.com'}]

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$

Okay, so there we've converted the original data structure to a new data structure. Once again,and I'm really trying to hammer this home, you need to get comfortable with this, dealing withthese complex data structure. You have to simplify the problem. Determine first if it's a list or a dictionary. If it's a list, look at the first element. Check the length.See how many elements it has. If it's a single element, just grab the single element. Remove thelist. If it's more than one element, then you're probably going to need to do a for loop and look atthe first entry. Remember to use your break so that you're only looking at the first entry.For the dictionary, you're going to want to start by looking at the keys. Then you're going to wantto look at each individual key, or if you see the key that's relevant to you, grab that key. Keepthis process of peeling back, peeling it back layer by layer. Then once you've figured out whatyou need to get to, start putting that in your code. You need to probably do that on an iterativemanner for these complex data structures. You figure out, I need to get element 0. Add that intoyour code. You figure out you need to get the interfaces key. Add that into your code. Keepsimplifying the problem step by step.

I am going to wrap some prints around this just to make it easier to read.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat data\_structures\_data\_format\_change.py

### data collected are not shown for easy reading###

from pprint import pprint

new\_list = []

for intf\_name, lldp\_info in lldp.items():

remote\_system = lldp\_info[0]

new\_entry = {

"local\_intf": intf\_name,

"remote\_intf": remote\_system['remote\_port'],

"remote\_system": remote\_system['remote\_system\_name'],

}

new\_list.append(new\_entry)

print()

pprint(new\_list)

print()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python data\_structures\_data\_format\_change.py

[{'local\_intf': 'Eth2/1',

'remote\_intf': 'Ethernet2/1',

'remote\_system': 'nxos2.twb-tech.com'},

{'local\_intf': 'Eth2/2',

'remote\_intf': 'Ethernet2/2',

'remote\_system': 'nxos2.twb-tech.com'},

{'local\_intf': 'Eth2/3',

'remote\_intf': 'Ethernet2/3',

'remote\_system': 'nxos2.twb-tech.com'},

{'local\_intf': 'Eth2/4',

'remote\_intf': 'Ethernet2/4',

'remote\_system': 'nxos2.twb-tech.com'}]

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$

Pyplus - Class3 - Serialization protocols

samedi, 23 mars 2019

11:03

let's start talking about YAML. The first question when we're talking about YAML,and this also relates to JSON, is why do we even need these things? Why do we even needsomething like YAML and something like JSON? We have this fundamental problem, that wehave things inside our computer program. In Python, this could be lists, and dictionaries, andcombination of these lists and dictionaries, and we also might have things like classes andobjects. We have these things inside of our program, and we have this issue of we might needto communicate those things from being inside of one computer to being inside of a differentcomputer, and these different computers could be on a totally different platforms. One of themcould be an Apple computer. One of them could be a Linux computer. One of them could beWindows, so they could be totally a dissimilar computers.We have this problem, that we have to take the things that are inside of our computer program,and then transfer those potentially across the network. This could also be potentially writingthem out to a file, and then reading them in later, either by the same program or by differentprograms. When we go through this process of transferring these things across the network, orwriting these things to a file, we need to take the objects that are inside of our program, andconvert them into a stream of bytes. This is known as this process of serialization. We're takingthe things that are inside of our program, and serializing them into a stream of bytes, either tobe written into a file or to be written across the network. We have to do it in a standardized way,such that the thing that reads them later, whether this is a remote computer or whether this isreading them later from a file, is able to understand them as what you created.

So we have this need for serialization protocols, and YAML and JSON are two of the commonserialization protocols. Now, what's the tradeoff of YAML versus JSON? Why would we useYAML instead of JSON?JSON is generally good for when you're doing computertocomputer communication. Whenhumans are not directly involved, it's very frequently the payload for APIs. A lot of webapplications you end up transferring JSON as the payload for the data. It also is good fornetworking device APIs. Things like Arista's eAPI will pass JSON data. Things like NXAPI, atleast in certain modes, will transfer JSON data. JSON is good for computertocomputerinteraction, but it's not that good if human beings have to actually write JSON. It's a bit picky and

it's fairly condensed, not super easy for human beings to write.So that's where YAML comes in. YAML has the advantage of that it's much easier for humanbeings to read and write it, at least in its expanded form. Things like Ansible, things like SALT, ifyou're representing an inventory of devices, these would be common uses for when you wouldactually be reading and writing, or when you'd be seeing YAML.Now, let's look into what YAML actually looks like. We'll start here and we'll make a simpleYAML file. Typically at the top of the YAML file, you'll have three hyphens. Now, in YAMLindentation matters.

It’s, I think, more finicky than Python is with respect to indentation, or more exactly, it's a bitharder to get your YAML indentation right, and it can be a bit more confusing when you mess itup, relative to Python.

How do we create a list in YAML? With a list, you just do a hyphen, space, and then we couldput whatever we want to put here for our list elements. So, we have router1, router2, router3,router4. We have a list that has four elements in it. Notice each one of these is hyphen, space.The thing over to the right is actually a string. In YAML, we don't need to quote our strings, sowe have four elements. Each one of them is a string. Then if I make this simple, relativelysimple, Python program, so what's this Python program do? I import the YAML Library. I promptfor a file name. I open that file name. I then just load whatever is in that file, but I load it asYAML, and then I print it out. Now if we run this, I'm going to run this with ‘ipython i’, the filename will be ‘test1.yml’.Now, notice what I get back here. That doesn't come in just as a string. This comes in as aPython list with four elements. We took the contents of what was in that YAML file, and wecreated a four element list from it. And if we do a ‘dir’, we can look at our YAML out variable,which is this. We can look at element zero. We can look at element one. We can do a ‘type’ onthis. So, this is not just a string.This comes in as a list, and comes in as a Python data structure that we can understand anddeal with directly.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat test1.yml

---

- rtr1

- rtr2

- sw1

- sw2

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat read\_yml\_in\_python.py

import yaml

### Filename is test1.yml###

filename = input("Enter filename:")

with open(filename) as f:

output = yaml.load(f)

print(output)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python read\_yml\_in\_python.py

Enter filename:test1.yml

['rtr1', 'rtr2', 'sw1', 'sw2']

Pyplus - Class3 - Yaml

samedi, 23 mars 2019

11:23

Let's look at writing YAML using Python. Here I import the YAML library. I create a dictionary,have some key-values. I create a list, I add that list to my dictionary, I add a null value to mydictionary. Then I add a Boolean to my dictionary. Now, here I say what the filename is, I just dothe standard Python that I'm going to write a file. Now, YAML has a condensed format and anexpanded format. In the expanded format, which is what I've been showing you, it looks at this expanded way. So, key-value. Now notice this came across as false, so lowercase. So that's the canonical representation that it's going to use for false. Here's our null value. Notice the stringsare not quoted here. Here's our list, here's all our individual entries of the list.I don't super love that this is not indented over, but that is actually proper list representation ofthat, so it knows that this key has to have a value associated with it. The value of this‘some\_list’, its corresponding value is this ten element list that you can see there.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat write\_yml\_in\_python.py

import yaml

my\_data = {

"rtr1": "10.1.1.1",

"device\_type": "cisco\_ios",

"username": "cisco",

"password": "cisco"

}

some\_list = list(range(5))

my\_data['some\_list'] = some\_list

my\_data['null\_value'] = None

my\_data['a\_boolean'] = True

filename = "yaml\_output.yml"

with open(filename, "wt") as f:

yaml.dump(my\_data, f, default\_flow\_style=False)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat yaml\_output.yml

a\_boolean: true

device\_type: cisco\_ios

null\_value: null

password: cisco

rtr1: 10.1.1.1

some\_list:

- 0

- 1

- 2

- 3

- 4

username: cisco

Now, what if we want to use the compressed format. We'll do ‘default\_flow\_style=True’. We'llrerun our program, let me resize this slightly so that it doesn't line wrap.Now, in compressed format, this starts to look a lot more like Python data structures, or likewhat JSON uses, and YAML is actually a superset of JSON, so JSON should be able to beunderstood by a YAML parser.But, notice in compressed format this looks a lot like what you see in Python. We have ourdictionary representation, now one thing you notice here is the keys, the strings, are not quoted.So, key, value, key, value, key, value. Boolean is represented as false, lowercase, null value'srepresented as ‘null’, and everything else looks fairly similar to what you would expect in Python.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat write\_yml\_in\_python.py

import yaml

my\_data = {

"rtr1": "10.1.1.1",

"device\_type": "cisco\_ios",

"username": "cisco",

"password": "cisco"

}

some\_list = list(range(5))

my\_data['some\_list'] = some\_list

my\_data['null\_value'] = None

my\_data['a\_boolean'] = True

filename = "yaml\_output.yml"

with open(filename, "wt") as f:

yaml.dump(my\_data, f, default\_flow\_style=True)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat yaml\_output.yml

{a\_boolean: true, device\_type: cisco\_ios, null\_value: null, password: cisco, rtr1: 10.1.1.1,

some\_list: [0, 1, 2, 3, 4], username: cisco}

Pyplus - Class3 - Json

samedi, 23 mars 2019

21:49

let's start talking about JSON. I already previously mentioned to you that JSON isfrequently used for computer-to-computer interaction. I mentioned previously that it's frequentlyused in web payloads. It's also frequently used for API, so for example, in Arista's eAPI, we'regoing to use JSON-RPC. In NX-API, in certain modes, we're going to use JSON for the payload.Now let's look at an example here, where we actually write out JSON. I import the JSONLibrary. I create a dictionary. I add a list to a dictionary. I add a null value and I add Boolean,very similar data structure to what we had in the YAML that we were writing out. Now here, I'mgoing to give it a file name file of ‘outfile.json’. Here I'm going to write out the file. Now withJSON, you could just dump it as the data you're going to dump.Here's the data we're going to dump. This ‘my\_data’ data structure and the file handle, the fileyou're going to write it to. You can also do this indent equals ‘indent=’, to make the JSON look alot prettier. That's a very nice trick to have in your tool belt, because when you get to large datastructures, if you don't do any indentation, it's going to be really hard to understand what thedata structure is.Now, this is going to write it out to a file, but I also want to just print it to the screen. Here I'mgoing to print, this ‘my\_data’ data structure, just in Python so that we can see it. Then downhere, I'm going to actually print it as JSON. Now notice when I print it here, I do the ‘.dump’, andI add the ‘s’ to it.This is going to do a ‘.dump’, and then the "s" means I'm going to dump it as a string.‘json.dump’ writes it to a file. ‘json.dumps’ with an "s" is going to write it out as a string. I pass in‘my\_data’ structure.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ nano write\_json\_in\_python.py

import json

my\_data = {

"rtr1": "10.1.1.1",

"device\_type": "cisco\_ios",

"username": "cisco",

"password": "cisco"

}

some\_list = list(range(5))

my\_data['some\_list'] = some\_list

my\_data['null\_value'] = None

my\_data['a\_boolean'] = True

filename = "json\_output.json"

with open(filename, "wt") as f:

json.dump(my\_data, f, indent=4)

At the output of this we're going to have an ‘json\_output.json’, we're going to write it to that file.We're going to also have a Python string representation of the data structure, and a JSON stringrepresentation of that data structure. Go ahead and execute this. Here we see, here's what itlooks like in Python. In Python, we have our dictionary representationWe have our stringsquoted in single quotes, and we know in Python, Python doesn't care about single quotes or adouble quotes. We get to something like our list. Here's our list represented as a squarebracket.We get to our null value. We know Python represents a null value as a none with a capital ‘N’.We get to our Boolean. We have a false with a capital ‘F’.Now let's compare that to what it looks like in JSON. JSON quotes it with double quotes. We getover here to our list. We saw our list, how square brackets, just like Python. Our dictionary hadcurly braces, just like Python. Now we get down to our ‘null\_value’ and we start to see somedifferences. Null value is ‘null’, just like it was with YAML, and our false is this lower case ‘f’.Once again, just like it was with YAML, but these are slightly different than what we had seen forPython.Now, if we look at our out ‘outfile.json’, now we can see this indent equals four, really helped ourrepresentation of it. We can see curly braces, curly braces, keys, values, keys, values, our list,write that all out, ‘null value: null’ and a Boolean at the end.Now, one thing that is picky about JSON is JSON does not like, and will generally cause issues,if you try to put a comma at the end of the last element. It's picky about this last element.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat json\_output.json

{

"rtr1": "10.1.1.1",

"device\_type": "cisco\_ios",

"username": "cisco",

"password": "cisco",

"some\_list": [

0,

1,

2,

3,

4

],

"null\_value": null,

"a\_boolean": true

}

Here let us compare the data structures in python vs json

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ nano write\_json\_in\_python.py

import json

my\_data = {

"rtr1": "10.1.1.1",

"device\_type": "cisco\_ios",

"username": "cisco",

"password": "cisco"

}

some\_list = list(range(5))

my\_data['some\_list'] = some\_list

my\_data['null\_value'] = None

my\_data['a\_boolean'] = True

filename = "json\_output.json"

with open(filename, "wt") as f:

json.dump(my\_data, f, indent=4)

### Print to stdout and compare python vs json representation ###

print('python')

print('#' \* 10)

print(my\_data)

print()

print('json')

print('#' \* 10)

print(json.dumps(my\_data))

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python write\_json\_in\_python.py

python

##########

{'rtr1': '10.1.1.1', 'device\_type': 'cisco\_ios', 'username': 'cisco', 'password': 'cisco', 'some\_list': [0, 1, 2, 3, 4], 'null\_value': None, 'a\_boolean': True}

json

##########

{"rtr1": "10.1.1.1", "device\_type": "cisco\_ios", "username": "cisco", "password": "cisco", "some\_list": [0, 1, 2, 3, 4], "null\_value": null, "a\_boolean": true}

Now we have this ‘json\_output.json’. Let's go ahead and try to read this in as Python. I import theJSON Library. I do pretty print. I'm going to just import that library. I prompt for a file name. Onceagain, I'm using Python 3 format here. I open that file name and then I just do a ‘json.load’. Ipass in my file handle. I'm going to get this data structure back, and then I'm just going to printthat, pretty print that, to the screen. The key point being there, I don't have just a string.I actually have a Python data structure. In fact, I have the exact same Python data structure thatI created in my original program, but I wrote it out to a file and then I re-read it back in later.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ python read\_json\_in\_python.py

Enter filename:json\_output.json

{'a\_boolean': True,

'device\_type': 'cisco\_ios',

'null\_value': None,

'password': 'cisco',

'rtr1': '10.1.1.1',

'some\_list': [0, 1, 2, 3, 4],

'username': 'cisco'}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ cat read\_json\_in\_python.py

import json

from pprint import pprint

### Filename is json\_output.json###

filename = input("Enter filename:")

with open(filename) as f:

output = json.load(f)

pprint(output)

Pyplus - Class3 - CiscoConfParse - part1

dimanche, 24 mars 2019

11:35

let's start talking about CiscoConfParse. CiscoConfParse is a library, and let's talka little bit about the context of when it's useful. We frequently, with Cisco and Cisco-like devices,we have situations where we have a hierarchy in the config, and that hierarchy is indicated byspace-based indentation. In Cisco, how something like an ‘interface FastEthernet4’, and weknow that these commands here are inside that context in the config. This IP address belongsto this ‘interface FastEthernet4’, and the duplex belongs to that ‘interface FastEthernet4’.

In Ciscoand Cisco-like devices, there is a hierarchy that's created, and that hierarchy is based on thesespace-based indentations. We could think of these as being global commands, and we couldthink of these internal commands as being children of this global command, they're inside thiscontext.Now, you could think that you could parse this in a certain way where you make a tree. In atree, these are the parent elements and these are the child elements to that parent. If you look at the at output of a cisco config from a router, there is a configuration SSH publickey chain [‘ip ssh pubkey-chain’].

Underneath that, I have a ‘key-hash’ associated with thatusername.This hierarchy can go to multiple levels of depth.Now some of these, if you tried to do this parsing with just using regular expressions, like if youtried to say something in like regular expressions, “I want to find the IP addresses that areunderneath ‘FastEthernet4’.” Some of those actually get to be fairly hard, fairly complex regularexpressions to do, or “I want to find the ‘bgp router-id’ that's under ‘router BGP 42’.”Some of those contexts where you're actually searching for a pattern that follows an existingpattern get to be pretty non-trivial with regular expressions. So having this tree parser capability is very nice, and that's what CiscoConfParse is going to do for us. It's going to parse the Ciscoconfig or Cisco-like configurations and convert it into a tree structure.

Let's look at an example here. I'm going to go into my ipython shell. I'm going to say ‘fromciscoconfparse import CiscoConfParse’. From this library, I'm going to import this class, andthen I'm going to provide in a file name. If I just provide in directly a file name, so here I'm goingto say ‘cisco1.txt’, and I'm going to save this to this ConfParse object [‘cisco\_obj’].

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class3]$ ipython

In [5]: from ciscoconfparse import CiscoConfParse

In [10]: CiscoConfParse("cisco1.txt")

Out[10]: <CiscoConfParse: 245 lines / syntax: ios / comment delimiter: '!' / factory: False>

You can see here that I have a ConfParse object. ‘CiscoConfParse’, it parsesd 245 lines. Itrecognizes its syntax as being IOS-like. It recognizes comments as being exclamation points.

Ijust want to look at what parsing this would look like if we just had straight config. Sometimesyou don't want to parse it as a file that you saved. You just want to pass in straight, a string thatyou've retrieved. Maybe you're directly connected to the device using Netmiko, you're executingshow run, and you don't actually want to save that to config.In ConfParse, if you are going to pass in a straight config. Let's say we had my config equalsthis, just as a string. There's a little gotcha here that can be annoying, confusing, when you areusing it, you can't just pass in this as a straight string. It's not going to like it as a straight string. See this

In [12]: my\_config = """

...: interface FastEthernet4

...: description \*\*\* LAN connection (don't change) \*\*\*

...: ip address 10.220.88.20 255.255.255.0

...: duplex auto

...: speed auto

...: """

In [14]: CiscoConfParse(my\_config)

[FATAL] CiscoConfParse could not open '

interface FastEthernet4

description \*\*\* LAN connection (don't change) \*\*\*

ip address 10.220.88.20 255.255.255.0

duplex auto

speed auto

'

---------------------------------------------------------------------------

FileNotFoundError Traceback (most recent call last)

~/VENV/py3\_venv/local/lib/python3.6/site-packages/ciscoconfparse/ciscoconfparse.py in \_\_init\_\_(self, config, comment, debug, factory, linesplit\_rgx, ignore\_blank\_lines, syntax)

217 config))

--> 218 f = open(config, \*\*self.openargs)

219 text = f.read()

FileNotFoundError: [Errno 2] No such file or directory: "\ninterface FastEthernet4\n description \*\*\* LAN connection (don't change) \*\*\*\n ip address 10.220.88.20 255.255.255.0\n duplex auto\n speed auto\n "

During handling of the above exception, another exception occurred:

RuntimeError Traceback (most recent call last)

<ipython-input-14-b7c4b805aac9> in <module>

----> 1 CiscoConfParse(my\_config)

~/VENV/py3\_venv/local/lib/python3.6/site-packages/ciscoconfparse/ciscoconfparse.py in \_\_init\_\_(self, config, comment, debug, factory, linesplit\_rgx, ignore\_blank\_lines, syntax)

285 except IOError:

286 print(("[FATAL] CiscoConfParse could not open '%s'" % config))

--> 287 raise RuntimeError

288 else:

289 raise RuntimeError("[FATAL] CiscoConfParse() received" +

RuntimeError:

What you need to do is convert this into a list.We pass in CiscoConfParse, we pass in‘my\_config’, and we call the ‘splitlines’ method. That's going to convert that ‘my\_config’ frombeing a string into being a list, and then CiscoConfParse can parse that correctly.

In [15]: CiscoConfParse(my\_config.splitlines())

Out[15]: CiscoConfParse: 5 lines / syntax: ios / comment delimiter: '!' / factory: False

Now the general things you want to use ConfParse for, we look at the ‘dir’ on this object, and ingeneral, we are going to want to couple this with some sort of ‘find\_objects’, so we are going tobe looking for certain structures in this that we care about.Let's first just look at this ‘help’ on ‘find\_objects’. It's going to take a ‘linespec’ and the ‘linespec’is going to be a regular expression pattern.

In [17]: type(cisco\_obj)

Out[17]: ciscoconfparse.ciscoconfparse.CiscoConfParse

In [16]: dir(cisco\_obj)

Out[16]:

['ConfigObjs',

'\_\_class\_\_',

'\_\_delattr\_\_',

'\_\_dict\_\_',

'\_\_dir\_\_',

'\_\_doc\_\_',

'\_\_eq\_\_',

'\_\_format\_\_',

'\_\_ge\_\_',

'\_\_getattribute\_\_',

'\_\_gt\_\_',

'\_\_hash\_\_',

'\_\_init\_\_',

'\_\_init\_subclass\_\_',

'\_\_le\_\_',

'\_\_lt\_\_',

'\_\_module\_\_',

'\_\_ne\_\_',

'\_\_new\_\_',

'\_\_reduce\_\_',

'\_\_reduce\_ex\_\_',

'\_\_repr\_\_',

'\_\_setattr\_\_',

'\_\_sizeof\_\_',

'\_\_str\_\_',

'\_\_subclasshook\_\_',

'\_\_weakref\_\_',

'\_build\_space\_tolerant\_regex',

'\_find\_all\_child\_OBJ',

'\_find\_line\_OBJ',

'\_find\_sibling\_OBJ',

'\_objects\_to\_uncfg',

'\_sequence\_nonparent\_lines',

'\_sequence\_parent\_lines',

'\_unique\_OBJ',

'append\_line',

'atomic',

'comment\_delimiter',

'commit',

'convert\_braces\_to\_ios',

'debug',

'delete\_lines',

'factory',

'find\_all\_children',

'find\_blocks',

'find\_children',

'find\_children\_w\_parents',

'find\_interface\_objects',

'find\_lineage',

'find\_lines',

'find\_objects',

'find\_objects\_dna',

'find\_objects\_w\_all\_children',

'find\_objects\_w\_child',

'find\_objects\_w\_missing\_children',

'find\_objects\_w\_parents',

'find\_objects\_wo\_child',

'find\_parents\_w\_child',

'find\_parents\_wo\_child',

'has\_line\_with',

'insert\_after',

'insert\_after\_child',

'insert\_before',

'ioscfg',

'objs',

'openargs',

'prepend\_line',

'replace\_all\_children',

'replace\_children',

'replace\_lines',

'req\_cfgspec\_all\_diff',

'req\_cfgspec\_excl\_diff',

'save\_as',

'sync\_diff',

'syntax']

In [20]: help(cisco\_obj.find\_objects)

Help on method find\_objects in module ciscoconfparse.ciscoconfparse:

find\_objects(linespec, exactmatch=False, ignore\_ws=False) method of ciscoconfparse.ciscoconfparse.CiscoConfParse instance

Let's say we wanted to do ‘cisco\_obj.find\_objects’and when you are doing regular expression patterns you always want to use raw string. You want to turn off the Python special characters and only have regular expression specialcharacters.

In [22]: cisco\_obj.find\_objects(r"interface")

Out[22]:

[IOSCfgLine # 171 'interface FastEthernet0',

IOSCfgLine # 174 'interface FastEthernet1',

IOSCfgLine # 177 'interface FastEthernet2',

IOSCfgLine # 180 'interface FastEthernet3',

IOSCfgLine # 183 'interface FastEthernet4',

IOSCfgLine # 189 'interface Vlan1']

Now, notice thisreturns a list like structure was returned, but if we start looking at a ‘dir’on this we can start to see something that's useful. Particularly when we go up to children

In [23]: intf = cisco\_obj.find\_objects(r"interface")

In [24]: dir(intf)

Out[24]:

['\_\_add\_\_',

'\_\_class\_\_',

'\_\_contains\_\_',

'\_\_delattr\_\_',

'\_\_delitem\_\_',

'\_\_dir\_\_',

'\_\_doc\_\_',

'\_\_eq\_\_',

'\_\_format\_\_',

'\_\_ge\_\_',

'\_\_getattribute\_\_',

'\_\_getitem\_\_',

'\_\_gt\_\_',

'\_\_hash\_\_',

'\_\_iadd\_\_',

'\_\_imul\_\_',

'\_\_init\_\_',

'\_\_init\_subclass\_\_',

'\_\_iter\_\_',

'\_\_le\_\_',

'\_\_len\_\_',

'\_\_lt\_\_',

'\_\_mul\_\_',

'\_\_ne\_\_',

'\_\_new\_\_',

'\_\_reduce\_\_',

'\_\_reduce\_ex\_\_',

'\_\_repr\_\_',

'\_\_reversed\_\_',

'\_\_rmul\_\_',

'\_\_setattr\_\_',

'\_\_setitem\_\_',

'\_\_sizeof\_\_',

'\_\_str\_\_',

'\_\_subclasshook\_\_',

'append',

'clear',

'copy',

'count',

'extend',

'index',

'insert',

'pop',

'remove',

'reverse',

'sort']

so let us check ‘interface sub-four’ [‘intf[4]’]

In [25]: intf[4]

Out[25]: <IOSCfgLine # 183 'interface FastEthernet4'>

In [26]: intf[4].children

Out[26]:

[IOSCfgLine # 184 ' description \*\*\* LAN connection (don't change) \*\*\*' (parent is # 183),

IOSCfgLine # 185 ' ip address 10.220.88.20 255.255.255.0' (parent is # 183),

IOSCfgLine # 186 ' duplex auto' (parent is # 183),

IOSCfgLine # 187 ' speed auto' (parent is # 183)]

It knows that IP address 10.2.20.88.20and this description and this duplex and the speed, that these are children of this parent line.We could do a loop over those. We have our parent line - our parent line is intf[4], and then we have our children line. We are starting to get something useful here. We did a regular expression search, we found theinterfaces, we can then see the children lines relative to each of the parents. The parent beingthe interface, the child being the things inside the config hierarchy that are underneath thoseinterfaces.

In [32]: for child in intf[4].children:

...: print(child.text)

...:

description \*\*\* LAN connection (don't change) \*\*\*

ip address 10.220.88.20 255.255.255.0

duplex auto

speed auto

In [34]: intf[4].text

Out[34]: 'interface FastEthernet4'

Pyplus - Class3 - CiscoConfParse - part2

dimanche, 24 mars 2019

12:17

let's continue onward with our discussion on CiscoConfParse. Remember we have aCiscoConfParse object. Let us try to find a parent which has a child that is matching specific string. For that we need to use "find\_objects\_w\_child" method.

I have parsed a cisco device output and assigned to the variable cisco\_obj.

In [76]: type(cisco\_obj)

Out[76]: ciscoconfparse.ciscoconfparse.CiscoConfParse

In [52]: cisco\_obj

Out[52]: <CiscoConfParse: 245 lines / syntax: ios / comment delimiter: '!' / factory: False>

In [53]: help(cisco\_obj.find\_objects\_w\_child)

find\_objects\_w\_child(parentspec, childspec, ignore\_ws=False) method of ciscoconfparse.ciscoconfparse.CiscoConfParse instance

Let us use this to find out a parent which has child that starts has a string with one or more white space followed ip address

In [54]: cisco\_obj.find\_objects\_w\_child(parentspec=r"interface",childspec=r"^\s+ip address")

Out[54]: [<IOSCfgLine # 183 'interface FastEthernet4'>]

Let us loop over and get the output of the parent and the child

In [72]: match = cisco\_obj.find\_objects\_w\_child(parentspec=r"interface",childspec=r"^\s+ip address")

In [73]: match

Out[73]: [<IOSCfgLine # 183 'interface FastEthernet4'>]

In [74]: from pprint import pprint

In [75]: for intf in match:

...: pprint(intf)

...: pprint(intf.children)

...:

<IOSCfgLine # 183 'interface FastEthernet4'>

[<IOSCfgLine # 184 ' description \*\*\* LAN connection (don't change) \*\*\*' (parent is # 183)>,

<IOSCfgLine # 185 ' ip address 10.220.88.20 255.255.255.0' (parent is # 183)>,

<IOSCfgLine # 186 ' duplex auto' (parent is # 183)>,

<IOSCfgLine # 187 ' speed auto' (parent is # 183)>]

Pyplus - Class3 - CiscoConfParse - Part3

dimanche, 24 mars 2019

21:35

Let's look at this example here with my ‘crypto map’, once you have a given parent. Let's do asearch on the following ‘cisco\_obj.find\_objects’ - going to do a search on that. We get backthose 5 lines, we say ‘match=’, we'll get one of these. Let's say we took the first line of that.Now once you have this IOS config line [IOSCfgLine], you can actually search inside of it. Wecould look at the children, but let's say we wanted to match the ‘set pfs’ inside of that. There's away you can actually search inside of the children. We could do ‘re\_search’, on the children[re\_search\_children] and we're going to do a regular expression search on the children line. Wecould do a further search inside of this and we could say, give me all the children of that, thathave ‘set pfs’ something. I only care about the lines inside of that, that have ‘set pfs’, and we doa further search.So you have this pretty nice capability of also searching on your results. So stepping back onceagain at a high level we know that Cisco config comes back as a tree hierarchy. We know thatactually dealing with that space-based tree hagiarchy is difficult from a regular expressionperspective. We know that we can use CiscoConfParse to parse these parent childrelationships. We know that with CiscoConfParse we can go from the parent object to the childobject. We also know that we can do searches in such a way that specify a parent and wespecify a child.

We know we can go from a child object back up to it's parent. We also know if we have a parentobject we can start searching on the children for additional patterns. In certain delimited context,this can be a helpful library to use and a helpful set of patterns to be aware of.

In [81]: new\_match = cisco\_obj.find\_objects(r"crypto map")

In [82]: new\_match

Out[82]:

[IOSCfgLine # 131 'crypto map CRYPTO 10 ipsec-isakmp ',

IOSCfgLine # 138 'crypto map CRYPTO 20 ipsec-isakmp ',

IOSCfgLine # 145 'crypto map CRYPTO 30 ipsec-isakmp ',

IOSCfgLine # 152 'crypto map CRYPTO 40 ipsec-isakmp ',

IOSCfgLine # 159 'crypto map CRYPTO 50 ipsec-isakmp ']

In [83]: new\_match[0].children

Out[83]:

[IOSCfgLine # 132 ' ! Incomplete' (parent is # 131),

IOSCfgLine # 133 ' set peer 1.1.1.1' (parent is # 131),

IOSCfgLine # 134 ' ! access-list has not been configured yet' (parent is # 131),

IOSCfgLine # 135 ' set transform-set AES-SHA ' (parent is # 131),

IOSCfgLine # 136 ' set pfs group5' (parent is # 131),

IOSCfgLine # 137 ' match address VPN-TEST1' (parent is # 131)]

In [88]: new\_match[0].re\_search\_children(r"set pfs")

Out[88]: [IOSCfgLine # 136 ' set pfs group5' (parent is # 131)]

Pyplus - Class4 - Python Libraries and PIP

lundi, 25 mars 2019

11:22

let's start talking about Python and libraries. Remember there are two fundamentalways in Python that you can import libraries. The first method is, you can just do it straight inimport on re ‘import re’. When you do an import of that case, you're specifying the name of thelibrary that you're importing. Then anything in your library that you want to use, like for example,‘re.search’, you would have to prefix the ‘re’ onto the name. This effects the naming of it insideyour program. Here you can see we have a reference to the ‘re.search’ method, we couldactually do a help on re.search ‘help(re.search) and find out more details about what we neededto do to actually use this search method, search function, inside the ‘re’ library.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ ipython

Python 3.6.7 (default, Dec 21 2018, 20:31:01)

Type 'copyright', 'credits' or 'license' for more information

IPython 7.2.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import re

In [2]: dir(re)

Out[2]:

['A',

'ASCII',

'DEBUG',

'DOTALL',

'I',

'IGNORECASE',

'L',

'LOCALE',

'M',

'MULTILINE',

'RegexFlag',

'S',

'Scanner',

'T',

'TEMPLATE',

'U',

'UNICODE',

'VERBOSE',

'X',

'\_MAXCACHE',

'\_\_all\_\_',

'\_\_builtins\_\_',

'\_\_cached\_\_',

'\_\_doc\_\_',

'\_\_file\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'\_\_spec\_\_',

'\_\_version\_\_',

'\_alphanum\_bytes',

'\_alphanum\_str',

'\_cache',

'\_compile',

'\_compile\_repl',

'\_expand',

'\_locale',

'\_pattern\_type',

'\_pickle',

'\_subx',

'compile',

'copyreg',

'enum',

'error',

'escape',

'findall',

'finditer',

'fullmatch',

'functools',

'match',

'purge',

'search',

'split',

'sre\_compile',

'sre\_parse',

'sub',

'subn',

'template']

In [3]: import json

In [4]: dir(json)

Out[4]:

['JSONDecodeError',

'JSONDecoder',

'JSONEncoder',

'\_\_all\_\_',

'\_\_author\_\_',

'\_\_builtins\_\_',

'\_\_cached\_\_',

'\_\_doc\_\_',

'\_\_file\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'\_\_path\_\_',

'\_\_spec\_\_',

'\_\_version\_\_',

'\_default\_decoder',

'\_default\_encoder',

'codecs',

'decoder',

'detect\_encoding',

'dump',

'dumps',

'encoder',

'load',

'loads',

'scanner']

Now, a second way that you can do an import is you can do a from library, import thing. So here,we're going to do a ‘from re import search’. Now this is going to mean that we don't prefix thiswith ‘re.search’. Instead, we're just going to call directly ‘search’. So we can directly justreference it as ‘search’, as opposed to ‘re.search’. Now that obviously implies that you don'thave anything else inside your program that's named ‘search’. So you know, you can't have aname conflict with this, with something else inside your programs or else that's going to causeproblems.

In [5]: from re import search

In [6]: search

Out[6]: <function re.search(pattern, string, flags=0)>

In [3]: from json import dump

In [7]: dump

Out[7]: <function json.dump(obj, fp, \*, skipkeys=False, ensure\_ascii=True, check\_circular=True, allow\_nan=True, cls=None, indent=None, separators=None, default=None, sort\_keys=False, \*\*kw)>

Now, you could also do the following: You could say ‘from re import search as’, whatever youwant to call this and that will make this be known as inside your program as this alternate name.So we have these two fundamental ways that we can import libraries

In [8]: from re import search as my\_search

In [9]: my\_search

Out[9]: <function re.search(pattern, string, flags=0)>

Now, there's a whole bunch of libraries that are included inside of Python. A large set of librariesthat are just directly available to you inside of Python. Now, there are additional libraries that areavailable to you using PIP, using ‘PIP’ and ‘pypi’. If we go to ‘pypi.python.org’, there's a bunch oflibraries available to us. We could search for Netmiko, and we could see that we have theNetmiko library available to us, the latest release as of this time is Netmiko 2.3.0.

How do you find out what libraries are available to you? You search on Google, you try to figureout what library is best, best matches is your particular use case. One thing you might want todo is look up the GitHub repository, get a sense of how active the project is, have there beencommits that have happened in the latest, you know, some interval of time. The latest twomonths, the latest four months. How many different users are contributing to it. Get a sense of how common, how widely used the project is, how active the developer is on working on thatproject.

Now if you want to install a given thing, we can just state ‘pip install netmiko’, and we can usePIP to install that package into our environment. You can also specify a particular version ofNetmiko, so if I wanted to PIP install Netmiko 2.3.0, I could do the following. Actually in thismachine, I actually previously had Netmiko 2.2.2, and now at this point in time, I'm going to haveNetmiko 2.3.0. PIP is definitely your friend, very, very helpful when you are trying to installpackages into your environment.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ pip install netmiko

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ pip install netmiko==2.3.0

Installing collected packages: netmiko

Found existing installation: netmiko 2.2.2

Uninstalling netmiko-2.2.2:

Successfully uninstalled netmiko-2.2.2

Successfully installed netmiko-2.3.0

If you want to see which packages you have at a given point in time, you can do PIP list, whichwill give you all the packages that you have installed. There's actually a lot of packages installedin this environment.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ pip list

Package Version

----------------- ----------

Django 1.11.17

dnspython 1.15.0

dnspython3 1.15.0

ipython 7.2.0

ipython-genutils 0.2.0

Jinja2 2.10

napalm 2.4.0

ncclient 0.6.3

netaddr 0.7.19

netmiko 2.3.0

nxapi-plumbing 0.5.2

paramiko 2.4.2

parso 0.3.1

pep8 1.7.1

pexpect 4.6.0

pickleshare 0.7.5

pip 18.1

pyeapi 0.8.2

pyflakes 2.0.0

pygal 2.4.0

Pygments 2.3.1

pyIOSXR 0.53

pylama 7.6.6

pylint 2.2.2

PyNaCl 1.3.0

pynetbox 4.0.6

pynxos 0.0.3

pyserial 3.4

pysmi 0.3.3

pysnmp 4.4.5

pytz 2018.7

PyYAML 3.13

scp 0.13.0

textfsm 0.4.1

You can actually do a ‘pip freeze’ if you want to do a different form of this. The PIP freeze formis very useful for actually recreating an environment. I could say ‘pip freeze’, which gives memy current state of all my packages.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ pip freeze

appdirs==1.4.3

asn1crypto==0.24.0

astroid==2.1.0

attrs==18.2.0

backcall==0.1.0

bcrypt==3.1.5

black==18.9b0

certifi==2018.11.29

cffi==1.11.5

chardet==3.0.4

ciscoconfparse==1.3.22

Click==7.0

colorama==0.4.1

cryptography==2.4.2

decorator==4.3.0

Django==1.11.17

dnspython==1.15.0

dnspython3==1.15.0

flake8==3.6.0

future==0.17.1

idna==2.8

ipaddress==1.0.22

ipdb==0.11

ipython==7.2.0

ipython-genutils==0.2.0

isort==4.3.4

jedi==0.13.2

Jinja2==2.10

jsonrpclib==0.1.7

junos-eznc==2.2.0

lazy-object-proxy==1.3.1

lxml==4.3.0

MarkupSafe==1.1.0

mccabe==0.6.1

napalm==2.4.0

ncclient==0.6.3

netaddr==0.7.19

netmiko==2.3.0

nxapi-plumbing==0.5.2

paramiko==2.4.2

parso==0.3.1

pep8==1.7.1

pexpect==4.6.0

pickleshare==0.7.5

ply==3.11

prompt-toolkit==2.0.7

ptyprocess==0.6.0

pyasn1==0.4.5

pycodestyle==2.4.0

pycparser==2.19

pycryptodomex==3.7.2

pydocstyle==3.0.0

pyeapi==0.8.2

pyflakes==2.0.0

pygal==2.4.0

Pygments==2.3.1

pyIOSXR==0.53

pylama==7.6.6

pylint==2.2.2

PyNaCl==1.3.0

pynetbox==4.0.6

pynxos==0.0.3

pyserial==3.4

pysmi==0.3.3

pysnmp==4.4.5

pytz==2018.7

PyYAML==3.13

requests==2.21.0

scp==0.13.0

selectors2==2.0.1

six==1.12.0

snowballstemmer==1.2.1

textfsm==0.4.1

toml==0.10.0

traitlets==4.3.2

typed-ast==1.1.1

urllib3==1.24.1

wcwidth==0.1.7

wrapt==1.10.11

xmltodict==0.11.0

I could then use "pip freeze" to create a ‘pypackage.txt’ file. I now have a file that corresponds to all the libraries and all the versions in my environment, and then I can actually do a ‘pip install r’. If I created a new virtual environment, that was totallyfresh, had nothing installed in it, I could reinstall these libraries with exactly these requirements,using this ‘pip install r’ option.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ pip freeze > pypackage.txt

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ ls

pypackage.tx

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ pip install r ./pypackage.txt

Pyplus - Class4 - sys.path and PYTHONPATH

lundi, 25 mars 2019

12:46

Continuing our discussion on Python libraries. Now we have this question of, “How does Pythonknow where to find something?” When it's searching to find a particular library, where does itsearch? How can we control that? If we go into Python and we import the Sys Module ‘importsys’, and I'm going to actually import PrettyPrint ‘from pprint import pprint’ as well. Then we do‘pprint(sys.path)’. Here you can actually see the paths that Python is going to search for. Thisnullstring is actually going to cause it to search for things in its current working directory.

Now notice a couple of these are actually my virtual environment, and ‘sitepackages’. It's worthnoting it's searching in ‘sitepackages’, and then here is a couple of more system locations thatit's looking for packages.

>>> import sys

>>> import pprint

>>> pprint(sys.path)

['',

'/home/sethuvignesh/python-libs',

'/home/sethuvignesh/DJANGOX/djproject',

'/home/sethuvignesh/VENV/py3\_venv/local/lib64/python3.6/site-packages',

'/home/sethuvignesh/VENV/py3\_venv/local/lib/python3.6/site-packages',

'/home/sethuvignesh/VENV/py3\_venv/lib64/python3.6',

'/home/sethuvignesh/VENV/py3\_venv/lib/python3.6',

'/home/sethuvignesh/VENV/py3\_venv/lib64/python3.6/site-packages',

'/home/sethuvignesh/VENV/py3\_venv/lib/python3.6/site-packages',

'/home/sethuvignesh/VENV/py3\_venv/lib64/python3.6/lib-dynload',

'/usr/lib64/python3.6',

'/usr/lib/python3.6']

Python is going to be looking when we do something like ‘import re’, it's going to be lookingthrough each one of these directories, trying to find this ‘re’ library in some way. If you want to see where it pulled it from, you could do ‘re\_\_.file\_\_’, and you could see in this particular case it got it from ‘lib64/python3.6/re.py’ module. We look up here and we find that, and we can seethat's actually /home/sethuvignesh/VENV/py3\_venv/lib64/python3.6 in this directory.

>>> import re

>>> re.\_\_file\_\_

'/home/sethuvignesh/VENV/py3\_venv/lib64/python3.6/re.py'

From my local windows machine.

>>> import sys

>>> pprint(sys.path)

['',

'C:\\Users\\vignesh.sethuraman\\python37.zip',

'C:\\Users\\vignesh.sethuraman\\DLLs',

'C:\\Users\\vignesh.sethuraman\\lib',

'C:\\Users\\vignesh.sethuraman',

'C:\\Users\\vignesh.sethuraman\\lib\\site-packages']

>>> import json

>>> json.\_\_file\_\_

'C:\\Users\\vignesh.sethuraman\\lib\\json\\\_\_init\_\_.py'

>>> import re

>>> re.\_\_file\_\_

'C:\\Users\\vignesh.sethuraman\\lib\\re.py'

At some point you're going to accidentally overwrite some system library. For example, you make a ‘netmiko.py’, you wonder why theNetmiko library no longer works. It's because its importing your ‘netmiko.py’ code, and it's nolonger importing the Netmiko library when you go to execute something. so be careful when naming a python script because it might overwrite an existing .py system library.

Now, an important point in Python is that you have to start to work out to get better at your programming is how do you reuse thing that you create? How do I take advantage of Python, toreuse things that I've created? So that you're not copying your code all over the place.

One way you can reuse something is just to make your own Python Module. Here I've made aPython Module, ‘my\_file.py’. It's in my current directory, so it's on my ‘sys.path’, right? We know‘sys.path’ that the first entry is the current working directory. So then I could just say, ‘import my\_file’. When I do ‘import my\_file’.

This is the first way of how to import the modules.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ nano my\_file.py

username = 'admin'

password = 'cisco'

>>> import my\_file

>>> my\_file.username

'admin'

>>> my\_file.password

'cisco'

>>> dir(my\_file)

['\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'password', 'username']

>>> my\_file.\_\_file\_\_

'/home/sethuvignesh/Viki/Class4/my\_file.py'

So we can put this anywhere on sys.path. Let us move it /home/sethuvignesh/VENV/py3\_venv/local/lib/python3.6/site-packages.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ mv my\_file.py ~/VENV/py3\_venv/local/lib/python3.6/site-packages

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ ls

\_\_pycache\_\_ pypackage.txt

The following is the second way of importing the modules. Here is we are importing only specific classes.

>>> from my\_file import username,password

>>> username

'admin'

>>> password

'cisco'

Notice you don't specify the ‘.py’, you just specify the module name. Where did it grab ‘my\_file’from? You can use the ‘\_\_.file\_\_’, well, I'd have to go ‘import my\_file’. So where did it grab thatfrom? it's grabbing it from this /home/sethuvignesh/VENV/py3\_venv/local/lib/python3.6/site-packages directory. It's somewhere on my 'sys.path'

>>> import my\_file

>>> my\_file.\_\_file\_\_

'/home/sethuvignesh/VENV/py3\_venv/local/lib/python3.6/site-packages/my\_file.py'

Now, if you want to modify your ‘sys.path’, so on Linux, Unix, you can set this ‘PYTHONPATH’variable.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ env | grep PYTHONPATH

PYTHONPATH=/home/sethuvignesh/python-libs:/home/sethuvignesh/DJANGOX/djproject/

you can set this ‘PYTHONPATH’variable, and that's why actually ‘nornir\_test’, ‘pythonlibs’, ‘DJANGOX/djproject’ etc, were shown up on my ‘sys.path'

It's important for you to be able to know, if you create some Python thing, how can I reuse that Python thing later? How can I tell Python to find it?. So this is why ‘sys.path’ is important. It's telling Python where tosearch, and you can control this in Linux, by using this ‘export PYTHONPATH’. There's alsoways you can control it in Windows as well,. You can tell Python, "How do I modify ‘sys.path’?,"and ‘sys.path’ is going to tell Python, where do I look for things that I want to reuse later.

Pyplus - Class4 - \_\_name\_\_ technique

lundi, 25 mars 2019

14:49

In this video, I want to talk to you about separating reusable code from executable code in Python.

You need to start learning how to create reusable things in your Python programs. There's atrick, a technique in Python that you're going to frequently see, which is this ‘if \_\_name\_\_ ==“\_\_main\_\_"’ technique.

This is just an "if" statement, ‘\_\_name\_\_’ is just an internal variable inPython that automatically gets set. We're saying if it equals a string “\_\_main\_\_”. I'm going to goahead and put ‘Executable code.

Generally the pattern you're going to follow is, you're going to put importable things up here,executable things down here. Now there's a question of well why the heck does this techniquework? Let's look at what this ‘\_\_name\_\_’ variable looks like. If we do Python on that, we'redirectly executing this Python script. We're going to see that variable ‘\_\_name\_\_’ is automatically set to this string ‘\_\_main\_\_’. But if we go in here and we actually ‘import test1.py’.We're going to see that this ‘\_\_name\_\_’ actually gets set in the import case to just be themodule name [test1].In the execution case it gets set to be the ‘\_\_main\_\_’ string. In the import case it gets set to the module name.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat test1.py

print(\_\_name\_\_)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python test1.py

\_\_main\_\_

In [1]: import test1

test1

The ‘\_\_name\_\_ ‘ variable gets set different ways depending on whether you import the Python file or whether you execute the Python file. We're going to take advantage of that to make atechnique where we can separate importable things in our code from executable things in ourcode.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat test1.py

print("Importable Code")

if \_\_name\_\_ == "\_\_main\_\_":

print("Executable Code")

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python test1.py

Importable Code

Executable Code

Pyplus - Class4 - Reusable code

vendredi, 29 mars 2019

17:15

I'm really trying to emphasize reusability and learning, knowing Python such that you can build things that you reuse in later programs this is a really big, important capability in programming. Not only are you accomplishing the thing that you intend to accomplish, but you're creating building blocks that you can use later in other programs and this starts to sort of compound your powers, compound your capabilities across time. Let's say for example, I create this file called ‘my\_devices.py’ and inside here I define a dictionary. I will define a dictionary.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat my\_devices.py

from getpass import getpass

password = getpass()

rtr3 = {

"host": 'cisco3.lasthop.io',

    "username": 'pyclass',

    "password": password,

    "device\_type": 'cisco\_ios'

}

rtr4 = {

"host": 'cisco4.lasthop.io',

    "username": 'pyclass',

    "password": password,

    "device\_type": 'cisco\_ios'

}

sw1 = {

"host": nxos1.lasthop.io',

    "username": 'pyclass',

    "password": password,

    "device\_type": 'cisco\_nxos'

}

sw2 = {

"host": nxos2.lasthop.io',

    "username": 'pyclass',

    "password": password,

    "device\_type": 'cisco\_nxos'

}

Now I could make a second program called this ‘netmiko\_connect.py’. What's the very first thing I'm going to do in this program? I'm going tosay ‘from my devices import rtr3, rtr4’sw1,sw2, whatever I named those particular devices. I can then create a second program that just imports those dictionaries. we actually verify that's working correctly. You can see I have this dictionary. I go into here. I could then do my standard Netmiko call. I can say ‘from netmiko import ConnectHandler’. I could then say I'm just going to connect to one of them, ‘net\_connect =ConnectHandler(\*\*rtr3)’, and I'll pass in ‘rtr3’ dictionary. Then I could do ‘print(net\_connect.find\_prompt()’.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat netmiko\_connect.py

from netmiko import ConnectHandler

from my\_devices import rtr3,rtr4,sw1,sw2

connect\_rtr3 = ConnectHandler(\*\*rtr3)

print (connect\_rtr3.find\_prompt())

connect\_rtr4 = ConnectHandler(\*\*rtr4)

print (connect\_rtr4.find\_prompt())

connect\_sw1 = ConnectHandler(\*\*sw1)

print (connect\_sw1.find\_prompt())

connect\_sw2 = ConnectHandler(\*\*sw2)

print (connect\_sw2.find\_prompt())

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python netmiko\_connect.py

Password:

cisco3#

cisco4#

nxos1#

nxos2#

I established an SSH Connection. I had a Python file, the ‘my\_devices.py’ that defined the device dictionaries, and then I actually created another program that imported things from that file. Remember, on ‘sys.path’, it’s always is going to look in your current directory, so as long as this ‘my\_devices.py’ is somewhere on this ’sys.path’, it'll be able to import this dictionary, it'll be able to import this dictionary. Then I just used that dictionary in this Netmiko class, I got my connection back, and I printed out the prompt. I have a Python module that I've now reused in another program.

You can reuse Python programs, Python code, from one program in another program. You have certain techniques available to you to do this.

You have your technique of doing the ‘\_\_name\_\_’, and, whether that equals ‘\_\_main\_\_’ or not, to separate importable code from executable code. We have our ‘sys.path’, as far as where Python is searching. We can create a new Python file, Python module, that has code that we want to reuse later. There are more advanced techniques where you can actually create a directory of related Python files that you can import, this is known as a Python package. So you're trying to build these building blocks of reusable things that you can use again in other Python programs.

Pyplus - Class4 TextFSM Overview

vendredi, 29 mars 2019

19:22

To set a bit of the context, let's starttalking about why we might need, might want to use something like TextFSM. TextFSM is aPython package that was originally created by Google. It's open source. The FSM part of thisstands for Finite State Machine.

One of the issues that we very frequently run into with networkautomation is that we're very frequently dealing with string text output from devices. There's a lotof devices out there in the field that do not have very good APIs. We're having to resort to theselegacy SSH method of just grabbing string output from devices and we need a way to parse thatstring output in devices.

Now one solution to this problem is to use Python, and to use regular expressions straight inPython. Now there's a set of problems with that and some of the problems with using that areone, your regular expressions get very complicated. They get very hard to maintain, they getreally hard to manage. Another important issue withusing these straight in Python is they become integrated and bound to your code. It is very hardto share these parsing definitions widely. Basically, your value that you get created is reallydirectly tied into your code and it's hard to use those things in other context easily betweendifferent projects.

Now some advantages of using it in Python. Python has good debugging and feedback tools.You can use the Python debugger, you can start printing statements out regarding what yourregular expressions are doing. Python regular expression engine is sort of widely known anddocumented, so those are some good things about using Python.

Now another solution we could use instead of using straight Python regular expressions is toinstead use this TextFSM module. Some of the advantages of using the TextFSM module isone, these are pretty easy to share. Two, having this finite state machine itself does give yousome meaningful values - in that some of these things that the finite state machine does areactually pretty hard to do straight in Python regular expressions. At the end of the day, youcould still accomplish them all, but they might not be as easy to solve straight in Python.

Pyplus - Class4 - TextFSM Structure and Template

vendredi, 29 mars 2019

23:08

One of the first things that we're going to do with TextFSM is we're actually going to definethese TextFSM template file. The template file is going to have this nature to it. At the very topof the template file, we're going to have these values that we define. These values are basicallygoing to be entries in the output that we're going to try to be extracting. The top of your TextFSMneeds to have a series of these values defined. The values need to be consecutive. You'lldefine them, one after the other, with no whitespace lines. You can have comments in betweenthem. The comments you can put in this way with your pound sign, but you won't have anywhitespace between your value definitions.Once you've defined these values, you'll then transition into the state machine part of yourtemplate. The state machine part of your template will have at least a separator whitespace line,and then you'll have your start field. You're always going to have a start with a capital ‘S’ foryour state machine start. Now the format of the values will always be ‘Value’, then you haveyour variable name.

By convention I'm going to always capitalize these variable names. I'm going to put them into allcapitals. You're then going to have a space, a parenthesis, and inside that parenthesis, you'regoing to put a regular expression pattern. This is going to be a regular expression pattern thatactually defines the meaning of, or the pattern that encompasses your variable name. I'll youshow you a bunch of examples of that. Hopefully that starts to make more sense. But it's goingto be a regular expression pattern that actually matches the meaning, the pattern thatencompasses that variable name.As I mentioned earlier, after all the values and variables are defined (well all the valuedefinitions are defined), you're going to put a whitespace line. You're then going to put your‘Start’ and you're going to start defining your state machine transition. Or you're going to startdefining your TextFSM state machine.

The second section in your TextFSM template is your state machine definition. The way you dotransitions in the state machine is this hyphen greater than [->] - you define a new state.

Now note one thing you need to be aware of is that there's ‘Implicit EOF; - an implicit end of fileinside TextFSM. You're template always has one of these built into it and it has an implicitRecord action. You can actually overwrite that behavior by having your own EOF. If you wantedto override the implicit EOF, you could put your own EOF state where you basically tell it to donothing. You could leave it just as I have right here, with just an EOF. That'll turn off the implicitEOF with a record.

let's start looking at how to install TextFSM and let's also look at an example of creating abasic template and using TextFSM.

I'm going to ‘pip install textfsm’, I'm going to also ‘pip install colorama’.

I'm going to use this toget some additional feedback out of TextFSM, so I can see what's going on better.Now if I do my ‘pip list’, I can see which packages I have installed. Now note,one thing to be aware of is there are three different TextFSM packages. This is sort of historicalreasons for why. Google wasn't maintaining it, so there was no Python 3 support. So a coupleother people created an alternative to TextFSM, but you're going to want to get the one that's maintained by Google - this is the one that in PyPI is just known as TextFSM. The currentversion, as of today, is 0.4.1.

Now, here is some ‘show\_ip\_int\_brief’ output that I have from a Cisco router. So pretty simpleoutput, it's clearly tabular output. Here we have an ‘Interface’, here we have an ‘IP-Address’, wehave a couple other fields, we have the line ‘Status’, the line ‘Protocol’.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat sh\_ip\_int\_brief.txt

Interface IP-Address OK? Method Status Protocol

FastEthernet0 unassigned YES unset down down

FastEthernet1 unassigned YES unset down down

FastEthernet2 unassigned YES unset down down

FastEthernet3 unassigned YES unset down down

FastEthernet4 10.220.88.20 YES NVRAM up up

Vlan1 unassigned YES unset down down

Let's look at the start of making a TextFSM template, and that processes this. So the very first thing I'm going to want to do is start creating my template, I'm just going to name it‘show\_ip\_int\_brief.template’. I'm going to start out by defining a value.

We'll do value and then I'm going to give this first value a name. I'm going to call it the‘INTERFACE’ Value, then I'm going to go parenthesis, and then I have to make a regularexpression that matches this interface definition. The regular expression that I'm going to use is‘\S+’, so non-whitespace repeated one or more times. Also let'sactually capture the IP-Address field. We'll do value, ‘IP\_ADDR’ which can either be unassigned or it can be an IP address. Now, the simple way to match on this is actually just going to be able to do our ‘\S+’ - non-whitespace one or more times. We go down here and we know that this isgoing to be our ‘INTERFACE’ whitespace, then we can say our ‘IP\_ADDR’ whitespace. Thatgets us here to here, all the way to here, all the way up to this. Our yes/no column which theseones are all a capital ‘S’. Let's go ahead and rerun our template and see if that worked properly

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat sh\_ip\_int\_brief.template

Value INTERFACE (\S+)

Value IP\_ADDR (\S+)

Start

^${INTERFACE}\s+${IP\_ADDR}\s+ -> Record

EOF

How do I execute this? I say Python, I call the ‘textfsm.py’ package, I then go and refer to my template and I then refer to my output that I'm parsing, the template I just created, and my showoutput that I've captured from my device. "python textfsm.py show\_ip\_int\_brief.templateshow\_ip\_int\_brief.txt"

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_ip\_int\_brief.template sh\_ip\_int\_brief.txt

FSM Template:

Value INTERFACE (\S+)

Value IP\_ADDR (\S+)

Start

^${INTERFACE}\s+${IP\_ADDR}\s+ -> Record

EOF

FSM Table:

['INTERFACE', 'IP\_ADDR']

['Interface', 'IP-Address']

['FastEthernet0', 'unassigned']

['FastEthernet1', 'unassigned']

['FastEthernet2', 'unassigned']

['FastEthernet3', 'unassigned']

['FastEthernet4', '10.220.88.20']

['Vlan1', 'unassigned']

Now, what if we want to remove this header row? Meaning the the row which has the headers "Interface, IP-Address, OK? Method, Status, Protocol" in sh\_ip\_int\_brief.txt file.If we want to remove that header row and let'sactually do this through introducing a state transition. We're going to get rid of this header row and we'll call this new state "intbief.

We'll go up to our start and we'll say whenever we encounter the line that begins with the literal string interface.\*, goes allthe way to protocol.\*. Now, I don't know if there's any whitespace at the end of this besides the

new line, so I'm going to put any consecutive whitespace, zero or more times and then I'm goingto do the end of line.Now, remember I said in regular expressions, end of line is indicated by a dollar sign. Butnotice, we're using dollar sign to indicate the special variable indicators. So there's a specialtrick we have to do to indicate an actual regular expression pattern of dollar sign. We actuallyhave to do dollar sign dollar sign [$$] to do the regular expression end of line.[^Interface.\*Protocol\s\*$$ ->].

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat sh\_ip\_int\_brief.template

Value INTERFACE (\S+)

Value IP\_ADDR (\S+)

Start

^Interface.\*Protocol\s\*$$ -> INTBRIEF

INTBRIEF

^${INTERFACE}\s+${IP\_ADDR}\s+ -> Record

EOF

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_ip\_int\_brief.template sh\_ip\_int\_brief.txt

FSM Template:

Value INTERFACE (\S+)

Value IP\_ADDR (\S+)

Start

^Interface.\*Protocol\s\*$$ -> INTBRIEF

INTBRIEF

^${INTERFACE}\s+${IP\_ADDR}\s+ -> Record

EOF

FSM Table:

['INTERFACE', 'IP\_ADDR']

['FastEthernet0', 'unassigned']

['FastEthernet1', 'unassigned']

['FastEthernet2', 'unassigned']

['FastEthernet3', 'unassigned']

['FastEthernet4', '10.220.88.20']

['Vlan1', 'unassigned']

At this point, we have the following. We're processing theInterface, and we're processing the IP-address. We also want to get the line Status and the lineProtocol.Notice in this particular example, we only have a down and an up value for each of these.There's also possibility of having administratively down and administratively up. Let's go aheadand just parse it for this current example. We'll do ‘LINE\_STATUS’, and here, we could say itcan either have a value of up, or it can have a value of down. We could also say line protocol[LINE\_PROTOCOL] could have a value of up or down.Then we go into here, and we replace this as opposed to saying some sequence of backslash splus a white space [\s+] - we could say any character repeated zero or more times. Then wecould do ‘LINE\_STATUS’, and we could do whitespace, and then ‘LINE\_PROTOCOL’ zero ormore whitespace at the end, end of line. Record that.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat sh\_ip\_int\_brief.template

Value INTERFACE (\S+)

Value IP\_ADDR (\S+)

Value LINE\_STATUS (up|down)

Value LINE\_PROTOCOL (up|down)

Start

^Interface.\*Protocol\s\*$$ -> INTBRIEF

INTBRIEF

^${INTERFACE}\s+${IP\_ADDR}.\*${LINE\_STATUS}\s+${LINE\_PROTOCOL}\s+$$ -> Record

EOF

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_ip\_int\_brief.template sh\_ip\_int\_brief.txt

FSM Template:

Value INTERFACE (\S+)

Value IP\_ADDR (\S+)

Value LINE\_STATUS (up|down)

Value LINE\_PROTOCOL (up|down)

Start

^Interface.\*Protocol\s\*$$ -> INTBRIEF

INTBRIEF

^${INTERFACE}\s+${IP\_ADDR}.\*${LINE\_STATUS}\s+${LINE\_PROTOCOL}\s+$$ -> Record

EOF

FSM Table:

['INTERFACE', 'IP\_ADDR', 'LINE\_STATUS', 'LINE\_PROTOCOL']

['FastEthernet0', 'unassigned', 'down', 'down']

['FastEthernet1', 'unassigned', 'down', 'down']

['FastEthernet2', 'unassigned', 'down', 'down']

['FastEthernet3', 'unassigned', 'down', 'down']

['FastEthernet4', '10.220.88.20', 'up', 'up']

Pyplus - Class4 - TextFSM Show Version Example

vendredi, 29 mars 2019

23:48

Here we show a little bit of a different use case because the output of ‘show\_version’ is nottabular output. It's not a table of rows like we saw with ‘show\_ip\_int\_brief’. It's a set of fields thatwe want to extract, and we're using TextFSM to accomplish that.

First let us try to pull the uptime and if that works lets iterate on other interesting options.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat sh\_version.template

Value UPTIME (\d\s\S+)

Start

^.\*uptime is ${UPTIME} -> Record

EOF

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_version.template sh\_version.txt

FSM Template:

Value UPTIME (\d\s\S+)

Start

^.\*uptime is ${UPTIME} -> Record

EOF

FSM Table:

['UPTIME']

['2 weeks,']

Now, let us pull the uptime as well as the image on the system.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat sh\_version.template

Value UPTIME (\d\s\S+)

Value IMAGE (.\*)

Start

^.\*uptime is ${UPTIME}

^System image file is ${IMAGE} -> Record

EOF

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_version.template sh\_version.txt

FSM Template:

Value UPTIME (\d\s\S+)

Value IMAGE (.\*)

Start

^.\*uptime is ${UPTIME}

^System image file is ${IMAGE} -> Record

EOF

FSM Table:

['UPTIME', 'IMAGE']

['2 weeks,', '"bootflash:c1100-universalk9\_ias.16.08.01.SPA.bin"']

Pyplus - Class4 - TextFSM Filldown-Fillup

lundi, 1 avril 2019

16:18

Let's look at the different output that I think, shows some other important ideas in TextFSM. Let's go ahead and look the‘show\_ip\_bgp.txt’ and notice that this is fairly interesting output. We have in this ‘show\_ip\_bgp’.This is actually been simplified, they were actually more entries before I edited it down. We haveat the top of this file, we have a set of sort of intro material. We then get down to the tabulardata and notice one interesting thing about the tabular data is that not all the fields always showup. For example, here, this network prefix is not included after the first entry. All these are stillreferring to ‘1.0.0.0/24’, but it's not repeated. Similarly, we get over here and there's some fieldsthat may or may not be there like Metric Local Preference ‘Metric LocPrf’. These kind of thingsstart to cause a lot of problems with regular expressions or start to cause issues with regularexpressions and this is a really good example of where the TextFSM state machine can be veryhelpful.Let's start looking at processing this output.

The very first thing I want to do is strip out this header material. We're going to have whitespacebeginning of the line. I want to basically find after this point because I want to just strip off theheader material and get to the tabular data. I'm going to have ‘Network’, some series ofcharacters and then I would have ‘Path’ and then potentially something at the end of the lineand then my double dollar sign [^\s+Network.\*Path\s\*$$]. Remember we have to put doubledollar sign because the dollar sign is a special character in TextFSM.Then, I'm going to transition to a different state. I'm going to transition to the ‘Info’. I'mgoing to transition to the tabular data. I'll go ahead and define this tabular data here. Now, let'ssay the first field we want to get is this star field. We look down here and we can see we canalso get star and we get greater than sign. I'm going to extract those as one entity and then Iwould post process these inside of Python if I wanted to separate them out. Star means it'svalid, greater sign means it's the best. So the best prefix, the best route. We'll call this field‘VALID\_BEST’. Now there could also be more combinations here. This is a classic example ofwhere you're going to need a large sampling of data to make sure you handle all the possiblecases, and this starts to get to be pretty hard problems. The larger your data set, the better offyou're going to be in your processing. We already know that this example is going to be asimplified example because we're not going to have other messages that could be included inthis first field.Now I'm going to make a set of characters, which I can indicate by brackets.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat sh\_ip\_bgp.template

Start

^\s+Network.\*Path$$ -> Info

It can either be star or it can be greater than sign ‘[\*>]+’. Now star is actually a regularexpression character, but since it's inside the character set (these square brackets) it’s justgoing to be treated as a literal star and a literal greater than sign. We're saying that it can beone of these two characters, star or greater than, and that one of those two characters has to berepeated one or more times. Now, we'll get down to here and we'll say start of the line, there's asingle space here. We'll go ahead and put that single space in. We'll go ahead and put ourvariable ‘VALID\_BEST’ in that and then we'll do dots.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_ip\_bgp.template sh\_ip\_bgp.txt

FSM Template:

Value VALID\_BEST ([\*>]+)

Start

^\s+Network.\*Path\s\*$$ -> Info

Info

^\s${VALID\_BEST} -> Record

EOF

FSM Table:

['VALID\_BEST']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*>']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*>']

['\*']

['\*']

['\*']

['\*']

['\*']

['\*']

Now, let's try to extract this prefix field. ‘PREFIX’ field, this is going to need to be an IP address,it's going to be a digit. Now, in regular expression, you can say the digit can happen from one tothree times. It can be there once or it can be there up to three times. This can be a value from,you know, one, two in the case of an IP prefix, something like 223 so we could have one, two,you know three, potential digits there.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_ip\_bgp.template sh\_ip\_bgp.txt

FSM Template:

Value VALID\_BEST ([\*>]+)

Value Filldown PREFIX (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}/\d{1,2})

Start

^\s+Network.\*Path\s\*$$ -> Info

Info

^\s${VALID\_BEST}\s+${PREFIX} -> Record

EOF

FSM Table:

['VALID\_BEST', 'PREFIX']

['\*', '1.0.0.0/24']

['\*', '1.0.4.0/24']

Note, here is where we have the overhanging. Which is going to start to be a problem, but fornow, let's just do it this way and let's see what happens here. Go ahead and execute it again.We can see that we actually get the two prefixes. That's all we have is that 1.0.0.0, 1.0.0.0/24and 1.0.4.0/24. But we missed all the other routes in the table, all the other next hops for thosetwo prefixes.

Let's go ahead and add the next hop field. The ‘NEXT\_HOP’ is going to be an IP address. It'lljust be like this except it won't have the prefix component. We'll go ahead and add the‘NEXT\_HOP’ in. Go ahead and do a ‘Record’.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_ip\_bgp.template sh\_ip\_bgp.txt

FSM Template:

Value VALID\_BEST ([\*>]+)

Value Filldown PREFIX (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}/\d{1,2})

Value NEXT\_HOP (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})

Start

^\s+Network.\*Path\s\*$$ -> Info

Info

^\s${VALID\_BEST}\s+${PREFIX}\s+${NEXT\_HOP} -> Record

EOF

FSM Table:

['VALID\_BEST', 'PREFIX', 'NEXT\_HOP']

['\*', '1.0.0.0/24', '208.74.64.40']

['\*', '1.0.4.0/24', '208.74.64.40']

Now, how do we solve this problem? The way we solve it is there's a way to do somethingcalled a ‘Filldown’. A ‘Filldown’ means that once I encounter this prefix, I'm going to save thisvariable for later use. Then, if I encounter a case where I don't have it later, I'm going to use thepreviously stored value. It's going to keep using this 1.0.0.0/24 until it's gets a new value whichis going to occur here. Now, one other thing we have to account for is in our rule processing.We have to account for the state when the prefix isn't there with the record operation.The first one handles the case when there's the VALID\_BEST’, the ‘PREFIX’, the ‘NEXT\_HOP’,it records it. But we also need a rule that says, also do a Record when it's just ‘VALID\_BEST’, abunch of whitespace and then ‘NEXT\_HOP’. Then our Filldown will cover the old prefix, theprefix that we've previously saved.We go and look at this. Now, I'm starting to get my tabular data including the prefix that's beenpreviously saved. In this case of where there are missing fields and you might need toremember them later, you're going to definitely want to start at using this Filldown.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm.py sh\_ip\_bgp.template sh\_ip\_bgp.txt

FSM Template:

Value VALID\_BEST ([\*>]+)

Value Filldown PREFIX (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}/\d{1,2})

Value NEXT\_HOP (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})

Start

^\s+Network.\*Path$$ -> Info

Info

^\s${VALID\_BEST}\s+${PREFIX}\s+${NEXT\_HOP} -> Record

^\s${VALID\_BEST}\s+${NEXT\_HOP} -> Record

EOF

FSM Table:

['VALID\_BEST', 'PREFIX', 'NEXT\_HOP']

['\*', '1.0.0.0/24', '208.74.64.40']

['\*', '1.0.0.0/24', '162.251.163.2']

['\*', '1.0.0.0/24', '94.142.247.3']

['\*', '1.0.0.0/24', '212.66.96.126']

['\*', '1.0.0.0/24', '91.218.184.60']

['\*', '1.0.0.0/24', '137.39.3.55']

['\*', '1.0.0.0/24', '132.198.255.253']

['\*', '1.0.0.0/24', '162.250.137.254']

['\*', '1.0.0.0/24', '208.51.134.254']

['\*>', '1.0.0.0/24', '202.232.0.2']

['\*', '1.0.4.0/24', '208.74.64.40']

['\*', '1.0.4.0/24', '209.124.176.223']

['\*', '1.0.4.0/24', '37.139.139.0']

['\*', '1.0.4.0/24', '12.0.1.63']

['\*', '1.0.4.0/24', '4.69.184.193']

['\*', '1.0.4.0/24', '195.208.112.161']

['\*', '1.0.4.0/24', '140.192.8.16']

['\*', '1.0.4.0/24', '198.58.198.254']

['\*', '1.0.4.0/24', '154.11.12.212']

['\*', '1.0.4.0/24', '202.93.8.242']

['\*', '1.0.4.0/24', '193.0.0.56']

['\*>', '1.0.4.0/24', '114.31.199.1']

['\*', '1.0.4.0/24', '134.222.87.1']

['\*', '1.0.4.0/24', '173.205.57.234']

['\*', '1.0.4.0/24', '198.58.198.255']

['\*', '1.0.4.0/24', '89.149.178.10']

['\*', '1.0.4.0/24', '203.62.252.83']

['\*', '1.0.4.0/24', '207.172.6.1']

Also, there is a concept if a Fillup. For example, if you have some data that you actually get atthe end of the show command output and you wanted to fill upwards in fields, you can also dothat. There are some other options that can come in here right between the value and the valuename definition

Pyplus - Class4 - TextFSM templates used in Python code

lundi, 1 avril 2019

16:48

I am going to show python code for every textfsm template that I used in previous exercises.

First let us look at the python code and how the sh\_ip\_int\_brief.template is used in that.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat textfsm\_intbrief.py

import textfsm

from pprint import pprint

template\_file = "sh\_ip\_int\_brief.template"

template = open(template\_file)

with open("sh\_ip\_int\_brief.txt") as f:

raw\_data = f.read()

read\_table = textfsm.TextFSM(template)

data = read\_table.ParseText(raw\_data)

pprint(data)

template.close()

Let us run the above code and see the output

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm\_intbrief.py

[['FastEthernet0', 'unassigned', 'down', 'down'],

['FastEthernet1', 'unassigned', 'down', 'down'],

['FastEthernet2', 'unassigned', 'down', 'down'],

['FastEthernet3', 'unassigned', 'down', 'down'],

['FastEthernet4', '10.220.88.20', 'up', 'up']]

Now, let let us look at the python code and how the sh\_version.template is used to grab the uptime and image from the show version ouput.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat textfsm\_uptimeandimage.py

import textfsm

from pprint import pprint

template\_file = "sh\_version.template"

template = open(template\_file)

with open("sh\_version.txt") as f:

raw\_data = f.read()

read\_table = textfsm.TextFSM(template)

data = read\_table.ParseText(raw\_data)

pprint(data)

template.close()

Let us run the above code and see the output

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm\_uptimeandimage.py

[['2 weeks,', '"bootflash:c1100-universalk9\_ias.16.08.01.SPA.bin"']]

Now, let let us look at the python code and how the sh\_ip\_bgp.template is used to grab the bgp best path,prefix and nexthop.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ cat textfsm\_filldown.py

import textfsm

from pprint import pprint

template\_file = "sh\_ip\_bgp.template"

template = open(template\_file)

with open("sh\_ip\_bgp.txt") as f:

raw\_data = f.read()

read\_table = textfsm.TextFSM(template)

data = read\_table.ParseText(raw\_data)

pprint(data)

template.close()

Let us run the above code and see the output

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class4]$ python textfsm\_filldown.py

[['\*', '1.0.0.0/24', '208.74.64.40'],

['\*', '1.0.0.0/24', '162.251.163.2'],

['\*', '1.0.0.0/24', '94.142.247.3'],

['\*', '1.0.0.0/24', '212.66.96.126'],

['\*', '1.0.0.0/24', '91.218.184.60'],

['\*', '1.0.0.0/24', '137.39.3.55'],

['\*', '1.0.0.0/24', '132.198.255.253'],

['\*', '1.0.0.0/24', '162.250.137.254'],

['\*', '1.0.0.0/24', '208.51.134.254'],

['\*>', '1.0.0.0/24', '202.232.0.2'],

['\*', '1.0.4.0/24', '208.74.64.40'],

['\*', '1.0.4.0/24', '209.124.176.223'],

['\*', '1.0.4.0/24', '37.139.139.0'],

['\*', '1.0.4.0/24', '12.0.1.63'],

['\*', '1.0.4.0/24', '4.69.184.193'],

['\*', '1.0.4.0/24', '195.208.112.161'],

['\*', '1.0.4.0/24', '140.192.8.16'],

['\*', '1.0.4.0/24', '198.58.198.254'],

['\*', '1.0.4.0/24', '154.11.12.212'],

['\*', '1.0.4.0/24', '202.93.8.242'],

['\*', '1.0.4.0/24', '193.0.0.56'],

['\*>', '1.0.4.0/24', '114.31.199.1'],

['\*', '1.0.4.0/24', '134.222.87.1'],

['\*', '1.0.4.0/24', '173.205.57.234'],

['\*', '1.0.4.0/24', '198.58.198.255'],

['\*', '1.0.4.0/24', '89.149.178.10'],

['\*', '1.0.4.0/24', '203.62.252.83'],

['\*', '1.0.4.0/24', '207.172.6.1']]

Pyplus - Class5 - **Jinja2 Variables** and Environments

lundi, 22 avril 2019

22:11

First let us install the Jinja2

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ pip install jinja2

Now let us build a basic bgp config template using Jinja2 templating format

I have my BGP configuration as template and dynamic values that are assigned in the config are passed as variables. Let us see the code and execute it

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_basics.py

from jinja2 import Template

bgp\_config = """

router bgp {{ bgp\_as }}

router-id {{ router\_id }}

neighbor {{ peer1 }} remote-as {{ peer1\_as }}

"""

j2\_template = Template(bgp\_config)

bgp\_cfg = j2\_template.render(bgp\_as="100",router\_id="10.220.88.20",peer1="10.220.88.30",peer1\_as="200")

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_basics.py

router bgp 100

router-id 10.220.88.20

neighbor 10.220.88.30 remote-as 200

Now let us expand by passing the variables using the \*\*keyword arguments instead of directly rendering it.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_basics.py

from jinja2 import Template

bgp\_config = """

router bgp {{ bgp\_as }}

router-id {{ router\_id }}

neighbor {{ peer1 }} remote-as {{ peer1\_as }}

"""

bgp\_vars = {

"bgp\_as":"100",

"router\_id":"10.220.88.20",

"peer1":"10.220.88.30",

"peer1\_as":"200"

}

j2\_template = Template(bgp\_config)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_basics.py

router bgp 100

router-id 10.220.88.20

neighbor 10.220.88.30 remote-as 200

In the above methods we were storing the variables and template directly on the python code. Now let try to move the template to a separate file and have the variables in the code.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat bgp\_config.j2

router bgp {{ bgp\_as }}

router-id {{ router\_id }}

neighbor {{ peer1 }} remote-as {{ peer1\_as }}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_codeonly.py

from jinja2 import Template

cfg\_file = "bgp\_config.j2"

with open(cfg\_file) as file\_lookup:

bgp\_template = file\_lookup.read()

bgp\_vars = {

"bgp\_as":"100",

"router\_id":"10.220.88.20",

"peer1":"10.220.88.30",

"peer1\_as":"200"

}

j2\_template = Template(bgp\_template)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_codeonly.py

router bgp 100

router-id 10.220.88.20

neighbor 10.220.88.30 remote-as 200

There are two generic problem with the above kind of approach. Problem number one is if we have a variable undefined. Here I'm undefining the ‘peer1’ variable. It's just going to silently work. It doesn't tell us that we're missing a variable, it just silently ignores the missing references and prints out the output. That, in this case, is very unlikely to be the behavior that we want.

Let me hide the variable peer1 and see what happens. .

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_codeonly.py

from jinja2 import Template

cfg\_file = "bgp\_config.j2"

with open(cfg\_file) as file\_lookup:

bgp\_template = file\_lookup.read()

bgp\_vars = {

"bgp\_as":"100",

"router\_id":"10.220.88.20",

# "peer1":"10.220.88.30",

"peer1\_as":"200"

}

j2\_template = Template(bgp\_template)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_codeonly.py

router bgp 100

router-id 10.220.88.20

**neighbor remote-as 200**

You can see the neighbor IP is missing from the config generated.

First of all, let's look at this. I say, ‘from jinja2.environment import Environment’ class. I then say, "Create a jinja2 environment," and I'm going to say, ‘undefined=StrictUndefined’. .

We create a jinja2 Environment. We then call ‘env.loader’, and we say, "Load the jinja2 templates, use this ‘FileSystemLoader’ class for where to find the template." Here we're just saying, "Look in the current working directory to try to find the template files.

Then I say, "Here are my variables for this template, here's my template file." Now I load it in a slightly different way. I don't read it in just using a Python file handle.

Instead, I use this ‘get\_template’, and I pass in the ‘template\_file’. There's my ‘template\_file’ that I'm passing in. It's using this Jinja2 Environment object, and this ‘get\_template’ method on that,and then I do my standard ‘template.render

Here is the code,

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_environ.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader(".")

bgp\_vars = {

"bgp\_as":"100",

"router\_id":"10.220.88.20",

# "peer1":"10.220.88.30",

"peer1\_as":"200"

}

cfg\_file = "bgp\_config.j2"

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_environ.py

Traceback (most recent call last):

**File "./bgp\_config.j2", line 3, in top-level template code**

**neighbor {{ peer1 }} remote-as {{ peer1\_as }}**

**jinja2.exceptions.UndefinedError: 'peer1' is undefined**

You can see the code has given us an error stating *peer1* variable is not defined. This means if a variable is not defined then it tell us that we're missing a variable, it do not just silently ignores the missing references and prints out the output.

Problem two, it doesn't nearly give us that much control, or at least we would have to start doing more

special file handling if we wanted to look in different directories for our templates. Right now we're reading from the current working directory, and we'd have to do special operations if we wanted to look for the template in a different paths.

Now I have created a directory called templates and moved the template file bgp\_config.j2 to the templates directory.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ mv /home/sethuvignesh/Viki/Class5/bgp\_config.j2 /home/sethuvignesh/Viki/Class5/templates/

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ ls

bgp\_config.j2

Now, when I run the python code, it should look for this file in the current working directory which is the directory class5.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ pwd

/home/sethuvignesh/Viki/Class5

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_environ.py

Traceback (most recent call last):

raise TemplateNotFound(template)

**jinja2.exceptions.TemplateNotFound: bgp\_config.j2**

So to address the problem 2, we have not modify the code to look for the files in a different paths as well. This FileSystemLoader will

"Look first in current working directory, then look in the ‘templates’ directory," and then it's going to look through each location one after the other trying to find the template file.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_environ.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

bgp\_vars = {

"bgp\_as":"100",

"router\_id":"10.220.88.20",

"peer1":"10.220.88.30",

"peer1\_as":"200"

}

cfg\_file = "bgp\_config.j2"

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_environ.py

router bgp 100

router-id 10.220.88.20

neighbor 10.220.88.30 remote-as 200

Pyplus - Class5 - Jinja2 Conditionals and whitespace stripping

lundi, 29 avril 2019

11:21

The Jinja2 conditionals aregoing to have a structure of ‘if’, ‘elif’, ‘else’, and then you always have to have this ‘endif’. Soyou have to have, at a minimum, you have to have an ‘if’ statement, and you have to have an‘endif’ statement, and they're going to use this ‘%}’ to indicate that it's a Jinja2 control structure.Here's the beginning of our ‘if’ statement, here's the end of our ‘if’ statement. We potentiallyhave the ‘elif’ and the ‘else’ statement. That's going to control whether these various stringblocks are included inside our template. So ‘if’ this section is true, we'll have this string. If the‘elif’ section is true we'll have this string. If the ‘else’ section, if both of these are false then we'llhave the ‘else’ section. These could be multiline strings. I've only shown a single line here, butthese could be multiline strings.

Now let's look at an example of this inside of Python. Here we have our Python2, Python3 gluecode. We have our Jinja2 glue code. We import our Environment class. We create anenvironment. We say we're going to enforce the undefined variables. If they are undefined,we're going to generate an exception. We specify our template loading that we're going tobasically load from the current working directory.We declare our variables and then we do our standard rendering using the Jinja2 environment.Now notice our template files is going to be this ‘intf\_config1.j2’.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_cond1.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

int\_vars = {"primary\_ip": True}

cfg\_file = "int\_config1.j2"

j2\_template = env.get\_template(cfg\_file)

int\_cfg = j2\_template.render(\*\*int\_vars)

print(int\_cfg)

Let's look at that. We lookat this and we see we have an ‘interface’. We have this ‘if primary\_ip’ is defined, we then have this ‘ip address’ string. In this example ifthis variable is true or treated by Python is true, we'll have this string. If it’s not true, then wewon't have that string.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cd templates/

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat int\_config1.j2

int gigabitethernet0/0

{% if primary\_ip %}

ip address 10.1.1.1 255.255.255.0

{% endif %}

negotiation auto

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_cond1.py

int gigabitethernet0/0

ip address 10.1.1.1 255.255.255.0

negotiation auto

Now notice that there is some whitespace issues that are going to come into play when this happens. Because there's a whitespace and there's a newline and there's a newline at the end of this. There'sgoing to be some whitespace issues when this is rendered.

The one that I use most commonly is to actually prefix various one of these curly brace percent,these block control structures with the hyphen. What the hyphen does is strip out leadingwhitespace. It's going to strip out that space there and it's going to strip out the newline.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat int\_config1.j2

int gigabitethernet0/0

{%- if primary\_ip %}

ip address 10.1.1.1 255.255.255.0

{%- endif %}

negotiation auto

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_cond1.py

int gigabitethernet0/0

ip address 10.1.1.1 255.255.255.0

negotiation auto

In general, you want to be pushing your complexity out of your template and into your Pythonprogram. This is a longstanding practice, or known practice in web templating. There's alongstanding rule in web templating where you push the complexity out of being in yourtemplate and move it into your program. We should be, in general, adopting that pattern ofcomplexity should be moved out of the template and into your Python program.Now, another thing you can do to help with your complexity of your template is use yourwhitespace stripping behavior to make your template more readable.

I'm just creating a Jinja2environment. I'm specifying some variables, and I'm pointing to a template

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_cond2.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

int\_vars = {"mode":"trunk"}

cfg\_file = "int\_config2.j2"

j2\_template = env.get\_template(cfg\_file)

int\_cfg = j2\_template.render(\*\*int\_vars)

print(int\_cfg)

Here is my template. I just have an‘interface Gi1/0/1’. I have an ‘if’ statement. I say ‘if mode == “access”’, I'm going to add these lines into my template. Then, I have an ‘elif’ statement and I say if ‘elif mode ==“trunk”’, I'm going to add these lines

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat int\_config2.j2

int gigabitethernet1/0/1

{%- if mode=="access" %}

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

{%- elif mode=="trunk" %}

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

{%- endif %}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_cond2.py

int gigabitethernet1/0/1

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

Now I change the value to access in the program and generate the template.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_cond2.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

int\_vars = {"mode":"access"}

cfg\_file = "int\_config2.j2"

j2\_template = env.get\_template(cfg\_file)

int\_cfg = j2\_template.render(\*\*int\_vars)

print(int\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_cond2.py

int gigabitethernet1/0/1

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

Here I'm just showing you an ‘if’/’else’ statement as opposed to an ‘if’/’elif’ statement. We look atmy Python program here. Here I've defined the mode to be null. So in that case, it's going toprint out the access mode. If I define that to be trunk, then it would print out the trunk section.There we see that our behavior that it renders is correct.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat int\_config2.j2

int gigabitethernet1/0/1

{%- if mode=="access" %}

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

{%- else %}

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

{%- endif %}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_cond2.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

int\_vars = {"mode":"access"}

cfg\_file = "int\_config2.j2"

j2\_template = env.get\_template(cfg\_file)

int\_cfg = j2\_template.render(\*\*int\_vars)

print(int\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_cond2.py

int gigabitethernet1/0/1

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_cond2.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

int\_vars = {"mode":"trunk"}

cfg\_file = "int\_config2.j2"

j2\_template = env.get\_template(cfg\_file)

int\_cfg = j2\_template.render(\*\*int\_vars)

print(int\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_cond2.py

int gigabitethernet1/0/1

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

Pyplus - Class5 - Jinja2 Nested Conditionals

lundi, 29 avril 2019

16:59

In this Iet us see the nested conditionals and how it is being used.

Here we have an “if” statement at the top, and we have an 'endif' at the bottom. It's basically saying, if this ‘bgp\_cfg’ is defined, then we're going to print the global bgp config section. And then inside that, you can see I have a "if" statement for ‘peer1’, and if peer1 is defined then inside that I'm going to activate it. If peer2 is defined then it's going to do this other ‘if’ statement. The 'endif' is going to be associated with this nearest ‘if’ statement. So we can have nested conditionals.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat templates/bgp\_config2.j2

{%- if bgp\_cfg is defined %}

router bgp {{ local\_as }}

router-id {{ router\_id }}

{%- if peer1 is defined %}

neighbor {{ peer1 }} remote-as {{ peer1\_as }}

address-family ipv4 unicast

neighbor {{ peer1 }} activate

exit

{%- endif %}

{%- if peer2 is defined %}

neighbor {{ peer2 }} remote-as {{ peer2\_as }}

address-family ipv4 unicast

neighbor {{ peer2 }} activate

exit

{%- endif %}

{%- endif %}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_nestcond.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

bgp\_vars = {

"bgp\_cfg":True,

"local\_as": "100",

"router\_id":"10.220.88.20",

"peer1":"10.220.88.30",

"peer1\_as":"220",

"peer2":"10.221.88.30",

"peer2\_as":"221"

}

cfg\_file = "bgp\_config2.j2"

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_nestcond.py

router bgp 100

router-id 10.220.88.20

neighbor 10.220.88.30 remote-as 220

address-family ipv4 unicast

neighbor 10.220.88.30 activate

exit

neighbor 10.221.88.30 remote-as 221

address-family ipv4 unicast

neighbor 10.221.88.30 activate

exit

Now I am going to see what happens when the first condition is not met. For example, if the bgp\_cfg is not defined, then you can see there was no config generated.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_nestcond.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

bgp\_vars = {

**# "bgp\_cfg":True**,

"local\_as": "100",

"router\_id":"10.220.88.20",

"peer1":"10.220.88.30",

"peer1\_as":"220",

"peer2":"10.221.88.30",

"peer2\_as":"221"

}

cfg\_file = "bgp\_config2.j2"

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_nestcond.py

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$

Now I am going to see what happens when the second condition is not met. For example, if the peer1 is not defined, then you can see there was no config generated for peer1 but rest of bgp global config and config related to peer2 is generated

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_nestcond.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

bgp\_vars = {

"bgp\_cfg":True,

"local\_as": "100",

"router\_id":"10.220.88.20",

**# "peer1":"10.220.88.30"**,

"peer1\_as":"220",

"peer2":"10.221.88.30",

"peer2\_as":"221"

}

cfg\_file = "bgp\_config2.j2"

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_nestcond.py

router bgp 100

router-id 10.220.88.20

neighbor 10.221.88.30 remote-as 221

address-family ipv4 unicast

neighbor 10.221.88.30 activate

exit

The point that I'm making here is that there's this conditional that defines whether a

variable is defined or not. So you can do this, “is variable defined?” and if it is or not, then that

whole section will be removed or added.

Pyplus - Class5 - Jinja2 Loops

lundi, 29 avril 2019

18:23

We see that we have a Jinja2 for loop and there's different forms that you can take as far as what you're looping over. Here I'm showing you looping over a list. It says ‘for item in some\_list’, then each time through the loop, this ‘item’ and variable is populated with some element in the list, one after the other, and then we can use that list in there as a variable inside of a string. I am going to configure a loop inside the if and else statements.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat int\_config3.j2

{%- if mode=="access" %}

{%- for intf\_number in range(1,4) %}

int gigabit1/0/{{ intf\_number }}

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

{%- endfor %}

{%- else %}

{%- for intf\_number in range(23,25) %}

int gigabit1/0/{{ intf\_number }}

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

{%- endfor %}

{%- endif %

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_loops1.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

int\_vars = {"**mode":"trunk"**}

cfg\_file = "int\_config3.j2"

j2\_template = env.get\_template(cfg\_file)

int\_cfg = j2\_template.render(\*\*int\_vars)

print(int\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_loops1.py

int gigabit1/0/23

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int gigabit1/0/24

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

Now if I change the mode to access, the access port config should be generated.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_loops1.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

int\_vars = {"mode":"access"}

cfg\_file = "int\_config3.j2"

j2\_template = env.get\_template(cfg\_file)

int\_cfg = j2\_template.render(\*\*int\_vars)

print(int\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_loops1.py

int gigabit1/0/1

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int gigabit1/0/2

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int gigabit1/0/3

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

Now, let us move the interface name into the program and assign into a variable and call them in the template. Also let us create a variable for switches in the stack, variable for the slot number and generate the config

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_loops2.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

intf\_list = []

for sw\_num in range(1,3):

base\_sw = f"Gigabit{sw\_num}/"

for slot\_num in range(1):

base\_intf = f"{base\_sw}{slot\_num}/"

for intf\_num in range(1,4):

intf\_name = f"{base\_intf}{intf\_num}"

intf\_list.append(intf\_name)

intf\_vars = {"intf\_list": intf\_list}

cfg\_file = "intf\_config4.j2"

j2\_template = env.get\_template(cfg\_file)

intf\_cfg = j2\_template.render(\*\*intf\_vars)

print(intf\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat intf\_config4.j2

{%- for intf\_number in intf\_list %}

int {{ intf\_number }}

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

{%- endfor %}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_loops2.py

int Gigabit1/0/1

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit1/0/2

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit1/0/3

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit2/0/1

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit2/0/2

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit2/0/3

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

Pyplus - Class5 - Jinja2 Loops - Enhanced

lundi, 29 avril 2019

23:55

I am going to create a program to generate the switch config for access ports and trunk ports.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_loops3.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

acc\_intf\_list = []

for sw\_num in range(1,3):

base\_sw = f"Gigabit{sw\_num}/"

for slot\_num in range(1):

base\_intf = f"{base\_sw}{slot\_num}/"

for intf\_num in range(1,4):

intf\_name = f"{base\_intf}{intf\_num}"

acc\_intf\_list.append(intf\_name)

trk\_intf\_list = []

for sw\_num in range(1,3):

base\_sw = f"Gigabit{sw\_num}/"

for slot\_num in range(1,2):

base\_intf = f"{base\_sw}{slot\_num}/"

for intf\_num in range(1,3):

intf\_name = f"{base\_intf}{intf\_num}"

trk\_intf\_list.append(intf\_name)

intf\_vars = {

"acc\_intf\_list": acc\_intf\_list,

"trk\_intf\_list": trk\_intf\_list

}

cfg\_file = "intf\_config5.j2"

j2\_template = env.get\_template(cfg\_file)

intf\_cfg = j2\_template.render(\*\*intf\_vars)

print(intf\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat intf\_config5.j2

{%- if acc\_intf\_list==acc\_intf\_list %}

{%- for acc\_intf\_number in acc\_intf\_list %}

int {{ acc\_intf\_number }}

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

{%- endfor %}

{%- endif %}

{%- if trk\_intf\_list==trk\_intf\_list %}

{%- for trk\_intf\_number in trk\_intf\_list %}

int {{ trk\_intf\_number }}

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

{%- endfor %}

{%- endif %}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_loops3.py

int Gigabit1/0/1

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit1/0/2

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit1/0/3

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit2/0/1

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit2/0/2

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit2/0/3

description "End-User Interface"

switchport mode access

switchport access vlan 300

switchport voice vlan 333

trust device cisco-phone

auto qos trust cos

spanning-tree portfast

spanning-tree bpduguard enable

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit1/1/1

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit1/1/2

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit2/1/1

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

int Gigabit2/1/2

switchport mode trunk

switchport trunk native vlan 400

udld port

trust cos

auto qos trust

service-policy input AutoQos-4.0-Trust-Cos-Input-Policy

service-policy output AutoQos-4.0-Output-Policy

!

Pyplus - Class5 - Jinja2 Lists

mercredi, 1 mai 2019

10:10

If you have a Jinja2 template, and you have a list embedded inside of it, you can do things like the following, where you basically refer to your list, and your access its indices. You can also loop over it using a for loop, like we have seen earlier.

Let's look at some examples of this. I have my standard code, it's very straightforward. We've seen this previously, basically, I'm just constructing a list of interfaces.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_lists.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

intf\_list = []

for sw\_num in range(1,3):

base\_sw = f"Gigabit{sw\_num}/"

for slot\_num in range(1):

base\_intf = f"{base\_sw}{slot\_num}/"

for intf\_num in range(1,4):

intf\_name = f"{base\_intf}{intf\_num}"

intf\_list.append(intf\_name)

intf\_vars = {"intf\_list": intf\_list}

cfg\_file = "list\_ex1.j2"

j2\_template = env.get\_template(cfg\_file)

intf\_cfg = j2\_template.render(\*\*intf\_vars)

print(intf\_cfg)

Now, my template, at this point, I am collecting the first element, second element, last element, all elements and length of the total elements in the list. For collecting the length, I used the jinja2 filter "length".

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat list\_ex1.j2

First intf:

{{ intf\_list[0] }}

Second intf:

{{ intf\_list[1] }}

Last intf:

{{ intf\_list[-1] }}

All intfs:

{{ intf\_list }}

Total intfs:

{{ intf\_list | length }}

Now I run the program and collect the outputs.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_lists.py

First intf:

Gigabit1/0/1

Second intf:

Gigabit1/0/2

Last intf:

Gigabit2/0/3

All intfs:

['Gigabit1/0/1', 'Gigabit1/0/2', 'Gigabit1/0/3', 'Gigabit2/0/1', 'Gigabit2/0/2', 'Gigabit2/0/3']

Total intfs:

6

Pyplus - Class5 - Jinja2 Dictionary

mercredi, 1 mai 2019

11:42

With Jinja2, you can refer to a dictionary, so you can do ‘{{ dict["key"] }}’, so that's your dictionary name and You can refer to its key. You can also do this dot notation so that can be convenient in certain context, so you could do ‘a\_dict.key’.

You can loop over dictionaries and just like with Python, by default, if you loop over a dictionary, you're going to get its keys back one after the other. Each time through this for loop, this ‘key’ variable which is just ... We could've called this X or whatever, but this variable named ‘key’, it's going to ‘key1’. We print that out, then it's going to get ‘key2’. We print out ‘key2’, ‘key3’, etc until we're all the way done with all our keys and then we'll just continue onward.

You can also loop through both the key and the values, using this ‘.items’ just like you can in a Python for loop. Each time through that, that's going to get you back a tuple. Here, we assign the key to the variable known as ‘k’. We assign the value to the variable known as ‘v’, and we print that out and we loop through those one after the other.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_dict.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

bgp\_vars = {

"local\_as":"100",

"router\_id":"10.100.88.30",

"peer1":"10.200.88.30",

"peer1\_as":"200"

}

bgp\_params = { "bgp\_vars": bgp\_vars }

cfg\_file = "dict\_ex1.j2"

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_params)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat dict\_ex1.j2

All Attributes:

{{ bgp\_vars.keys() }}

Local\_AS:

{{ bgp\_vars["local\_as"] }}

BGP Router\_ID:

{{ bgp\_vars.router\_id }}

Peer1\_IP and Peer1\_AS:

{{ bgp\_vars["peer1"] }} and {{ bgp\_vars["peer1\_as"] }}

Using Loops to print keys only:

{%- for attributes in bgp\_vars %}

{{ attributes }}

{%- endfor %}

Using Loops to print keys and values:

{%- for attributes,values in bgp\_vars.items() %}

{{ attributes }} - {{ values }}

{%- endfor %}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_dict.py

Local\_AS:

100

BGP Router\_ID:

10.100.88.30

Peer1\_IP and Peer1\_AS:

10.200.88.30 and 200

Using Loops to print keys only:

local\_as

router\_id

peer1

peer1\_as

Using Loops to print keys and values:

local\_as - 100

router\_id - 10.100.88.30

peer1 - 10.200.88.30

peer1\_as - 200

Pyplus - Class5 - Jinja2 Create Variables and Filters

mercredi, 1 mai 2019

12:54

let's look at how we create a variable inside a Jinja2 template. I have this Jinja2

template right here. I'm just going to create this variable and I'm going to set it to a

Value.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat filters\_ex1.j2

{%- set local\_as = "100" %}

{%- set router\_id = "10.100.88.30" %}

{%- set peer1 = "10.200.88.30" %}

{%- set peer1\_as = "200" %}

{{ local\_as }}

{{ router\_id }}

{{ peer1 }}

{{ peer1\_as }}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_filters.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

bgp\_vars = {}

bgp\_params = { "bgp\_vars": bgp\_vars }

cfg\_file = "filters\_ex1.j2"

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_params)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_filters.py

100

10.100.88.30

10.200.88.30

200

Notice what I do in this first Jinja2 expression. I say ‘var1’ and then I pipe it through the

‘capitalize’. This is going to be a string operation, going to pipe it through ‘capitalize’. It's going to capitalize this leading "l" and convert this from being all lowercase to being uppercase L using that capitalize filter. This pipe symbol is indicating we're going to do a Jinja2 filter and here's the filter that we're referring to. We also see if we can convert the whole string to uppercase [upper]. We convert it entirely to uppercase.

Then we're going to center it in a row that's is specified. The value specified in the center () depends on how much spacing needed.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat filters\_ex1.j2

{%- set bgp\_intf = "loopback0" %}

{%- set local\_as = "100" %}

{%- set router\_id = "10.100.88.30" %}

{%- set peer1 = "10.200.88.30" %}

{%- set peer1\_as = "200" %}

{{ bgp\_intf | capitalize }}

{{ bgp\_intf | upper }}

{{ bgp\_intf | center(16) }}

{{ local\_as | center(10) }}

{{ router\_id | center(18) }}

{{ peer1 | center(18) }}

{{ peer1\_as | center(10) }}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_filters.py

Loopback0

LOOPBACK0

loopback0

100

10.100.88.30

10.200.88.30

200

Jinja2 filters are heavily used in Ansible and they're pretty important concept in an Ansible context. Ansible has a set of special Jinja2 filters that were created just for it.

You can go and look at the documentation on Jinja2 filters to see what Jinja2 filters are available to you. They can be useful in some context. Once again, I really want to hammer on the point how we are to be careful about putting too much complexity in to your Jinja2 templates. Both with these set constructs and also with these filter constructs. Use this whitespace stripping behavior to make your Jinja2 templates more readable if you do have these constructs inside of them.

Pyplus - Class5 - Jinja2 template inheritance

mercredi, 1 mai 2019

17:25

I am going to do template inheritance using the "include" jinja2 method. This is method of statically assigning a template inside another template. For example, in this example I am statically assigning the "filters\_ex1.j2" inside a "include\_ex1.j2". I am referencing the outer template "include\_ex1.j2" template in the python code.

Let's think of some examples of where we could use this. Think, for example, if you're using Jinja2 for configuration templating, you could have your big configuration file, and maybe initially you just have certain sections where you're using more global variables that you can pull in. For example, you could pull in your SNMP configuration. You could pull in your AAA configuration section.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat include\_ex1.j2

{# include the template from another template #}

{%- include "filters\_ex1.j2" %}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat filters\_ex1.j2

{%- set bgp\_intf = "loopback0" %}

{%- set local\_as = "100" %}

{%- set router\_id = "10.100.88.30" %}

{%- set peer1 = "10.200.88.30" %}

{%- set peer1\_as = "200" %}

{{ bgp\_intf | capitalize }}

{{ bgp\_intf | upper }}

{{ bgp\_intf | center(16) }}

{{ local\_as | center(10) }}

{{ router\_id | center(18) }}

{{ peer1 | center(18) }}

{{ peer1\_as | center(10) }}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_include.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

bgp\_vars = {}

cfg\_file = "include\_ex1.j2"

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_include.py

Loopback0

LOOPBACK0

loopback0

100

10.100.88.30

10.200.88.30

200

The second way of doing the template inheritance is assigning the template to a variable in the python code and referencing the variable in another template. For example, I am assigning the "filters\_ex1.j2" template to a variable "dynamic\_template". This variable is referenced in my outer jinja2 template "include\_ex2.py". Based on the need, the variable "dynamic\_template" can be assigned a different template. This means the outer template is constant but the template assigned to the variable is dynamic.

You could think of some examples where this might be useful. Say you have this configuration templating system that you're doing, and you have some set of information that's basically identical. For example, say in the Cisco world, we have a bunch of global parameters that are the same, and a bunch of footer parameters that are the same.

But then in the middle, we have this interface section that varies by model, and maybe the

interface names change, and maybe how the interfaces are configured are changed. You could potentially create different templates for different models. You could say if it's model X, I'm going to load the interface section from this one template. If it's model Y, I'm going to load the interface section from this other template. Based on the value of some variable, you start loading in these other templates to complete this configuration templating.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ cat jinja2\_include.py

from \_\_future\_\_ import unicode\_literals, print\_function

from jinja2 import FileSystemLoader, StrictUndefined

from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)

env.loader = FileSystemLoader([".","./templates"])

bgp\_vars = {**"dynamic\_template": "filters\_ex1.j2"**}

cfg\_file = **"include\_ex2.j2"**

j2\_template = env.get\_template(cfg\_file)

bgp\_cfg = j2\_template.render(\*\*bgp\_vars)

print(bgp\_cfg)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 templates]$ cat include\_ex2.j2

{# Assign another template dynamically using a variable#}

{%- include dynamic\_template %}

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class5]$ python jinja2\_include.py

Loopback0

LOOPBACK0

loopback0

100

10.100.88.30

10.200.88.30

200

There are a few other advanced Jinja2 topics that I just want to make reference to the fact that these things exists and they might, in certain situations, be worth you knowing about. Jinja2 has a way of doing functions. They are known as macros in Jinja2. They operate in a very similar manner to Python functions. You basically call these macros and they expand a section of text that you use in the place you use where you called them from. You could have some macro that you called in multiple places if you have a repeating block of text that you wanted use throughout whatever you’re generating from your configuration template. Jinja2 has functions and they are known as macros.

There also is, in Jinja2, a second way of doing what I’ll call template inheritance. I’ve showed you earlier a way of including one Jinja2 file inside another one. You can have a block that you’re grabbing from a second Jinja2 file and pulling in to your main file. There’s this second way of doing this composition, and it’s known as Jinja2 template inheritance. It gets a little bit complicated, but you can think of it as a way of composing a rendered output from a set of files that have defined relationships between them. Basically, you compose an output in some way with multiple files and there’s relationships between the files. There’s certain Jinja2 keywords that come into play here like, ‘extend’, and ‘blocks’ and basically you’re inserting these ‘blocks’ into this ultimately based template you’re building upon. I’ve found this template inheritance to be useful in web templating. I’ve definitely use that with rendering web pages and the output there.

In general with network configuration templating, I have found the include paradigm more

straightforward to use, and easier to use. In general, I have used that include pattern more often in the network configuration templating. But it is probably worth while for you to know there is a second way of composing from multiple Jinja2 templates called template inheritance, and in certain situations, it might be useful.

Pyplus - REST API Introduction and Characteristics

mercredi, 1 mai 2019

18:41

The first question that we have here is, why as network engineers should we even care about REST API’s?

The general answer is, we have several things out there that we might be using in one way or another that are going to have a REST API and we need to know how, we need to learn how to programmatically interface to those things. For example, a lot of times, IP address management tools have REST API’s. Or data center infrastructure management tools have REST API’s. Things like Infoblox, NetBox, Network Source of Truth, these type of tools will have a REST API that you might need to programmatically interface to. Another category of things that have REST API’s that we might, as network engineers, need to use are controllers of one form or another. So things like the big switch SDN controller, the APIC from Cisco, NSO from Cisco. There might be some sort of centralized controller that has a REST interface that we need to programmatically interface to. Another category of things that might have a REST API that we need to use is actual devices themselves. Things like the Cisco ASA and their REST API, it’s REST API. Then finally the last category of things that have REST API’s very frequently is internet services. We might need to programmatically interface to some set of internet services for some sort of a workflow. Things like GitHub, and there are a lot of services out in the internet that have a RESTAPI available to them, like credit card processing, project management processing, other productivity tools, and other applications that could be relevant to you in some way. So things that might be in your CICD pipeline might have a REST API into them. Basically, a lot of internet services have REST APIs.

REST API’s - The term REST stands for Representational State Transfer.

In general, there's going to be four fundamental operations we can do. We can do a retrieve the resource. In HTTP terms that's going to a GET. We can do some sort of a POST of the resource, so this is a creating of an object. We could do a PUT of the resource. The PUT of the resource is going to be a modification of an existing resource. Then a DELETE operation is going to be delete, obviously remove that resource. This is also frequently known in terms of if you have a database application, these are also frequently known as CRUD applications, CRUD, where the C stands for create. The R stands for read. The U stands for update. The D stands for delete. Create, read, update, delete, so these four fundamental operations, and we have corresponding HTTP operations that we can use to accomplish those. You have these “nouns”, which are your HTTP URLs, and then you have these “verbs”, these are these actions (get, post, put, delete) that we can do on those nouns. We're going to be doing those all through our resource specifier. In general, when you're interfacing to a REST API, you're basically going to be changing these URLs that you access. Then you're going to be doing certain operations on those URLs. The majority of the time these are very, very frequently either GET operations or POST operations. These could be other things we do, but a lot of the time it's going to be either a GET operations or a POST operation.

Pyplus - REST API GET from Browser​

mercredi, 1 mai 2019

19:11

let's start talking abouthow do we programmatically interface to a REST API? The first part of that process is, how dowe start gathering information about this REST API so that we know how to use it?One, we need to start looking at the API documentation. Two, we need to start looking if there'sany Python. If there are any Python libraries available to us, that can help us programmaticallyinterface to it.Another thing that's very helpful with REST APIs is, since they're based around HTTP, we canstart to use standard utilities like our browser and some other command line utilities to actuallystart to use those utilities to gather information about the API. Here we're just using our browser.We can start to traverse things in the API using our browsers and to start to figure out thingsthat are available to us that we might want to use.We can also start to figure out what we might need to construct in our program to use these APIs.

From the command line you can use 'curl’.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Viki]$ curl -s <https://netbox.lasthop.io/api/> --insecure | jq "."

{

  "circuits": "<http://netbox.lasthop.io/api/circuits/>",

  "dcim": "<http://netbox.lasthop.io/api/dcim/>",

  "extras": "<http://netbox.lasthop.io/api/extras/>",

  "ipam": "<http://netbox.lasthop.io/api/ipam/>",

  "secrets": "<http://netbox.lasthop.io/api/secrets/>",

  "tenancy": "<http://netbox.lasthop.io/api/tenancy/>",

  "virtualization": "<http://netbox.lasthop.io/api/virtualization/>"

}

I'm using thisinsecure option because I don't want it to actually validate the SSL certificate on this. I'maccessing this URL here. I'm using the ‘s’ option to eliminate some of the header, some of theother information that cURL returns. Then I'm going to do this ‘jq’ operation just to make the JSON print out in a prettier format.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Viki]$ curl -s <https://netbox.lasthop.io/api/dcim/devices/> --insecure | jq "."

{

  "detail": "Authentication credentials were not provided."

}

Now once I get into the devices, I actually transition into things that start needing authentication.With NetBox, I actually need an authorized token that I need to provide to give additionalinformation. You can see that I'm using these command line browser based tools to get information.

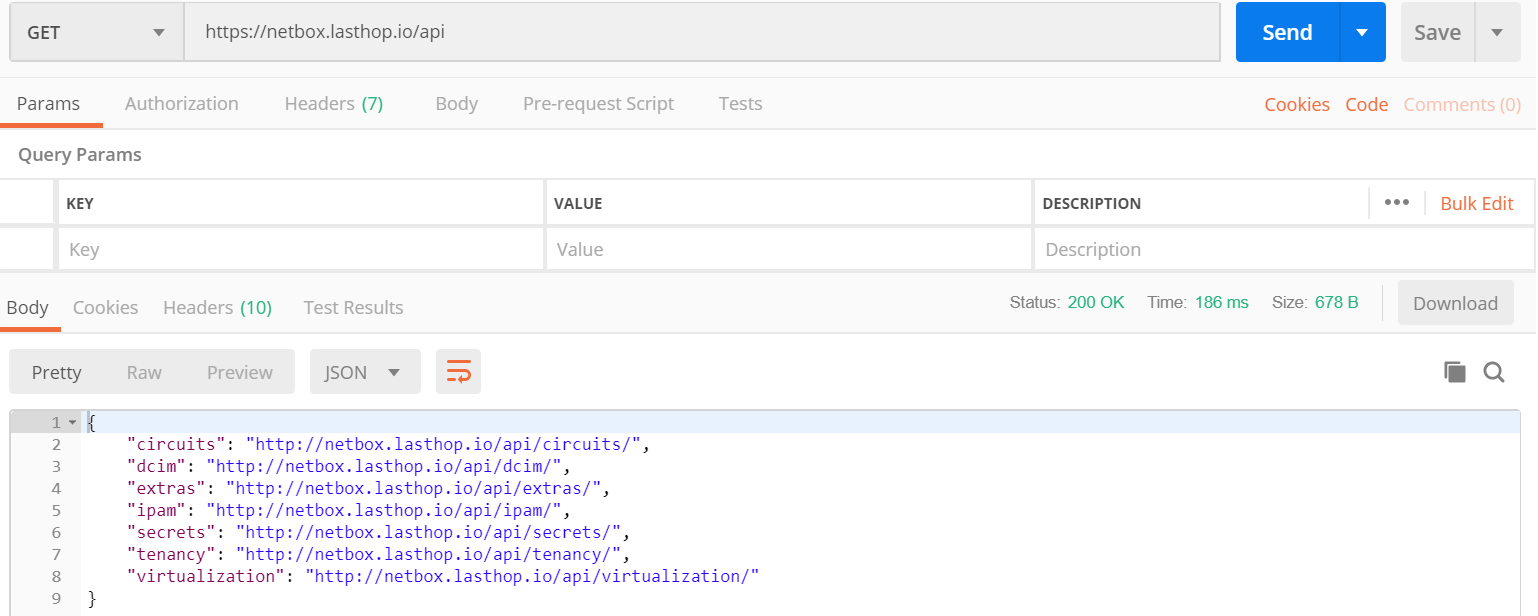
Pyplus - REST API GET from Postman​

mercredi, 1 mai 2019

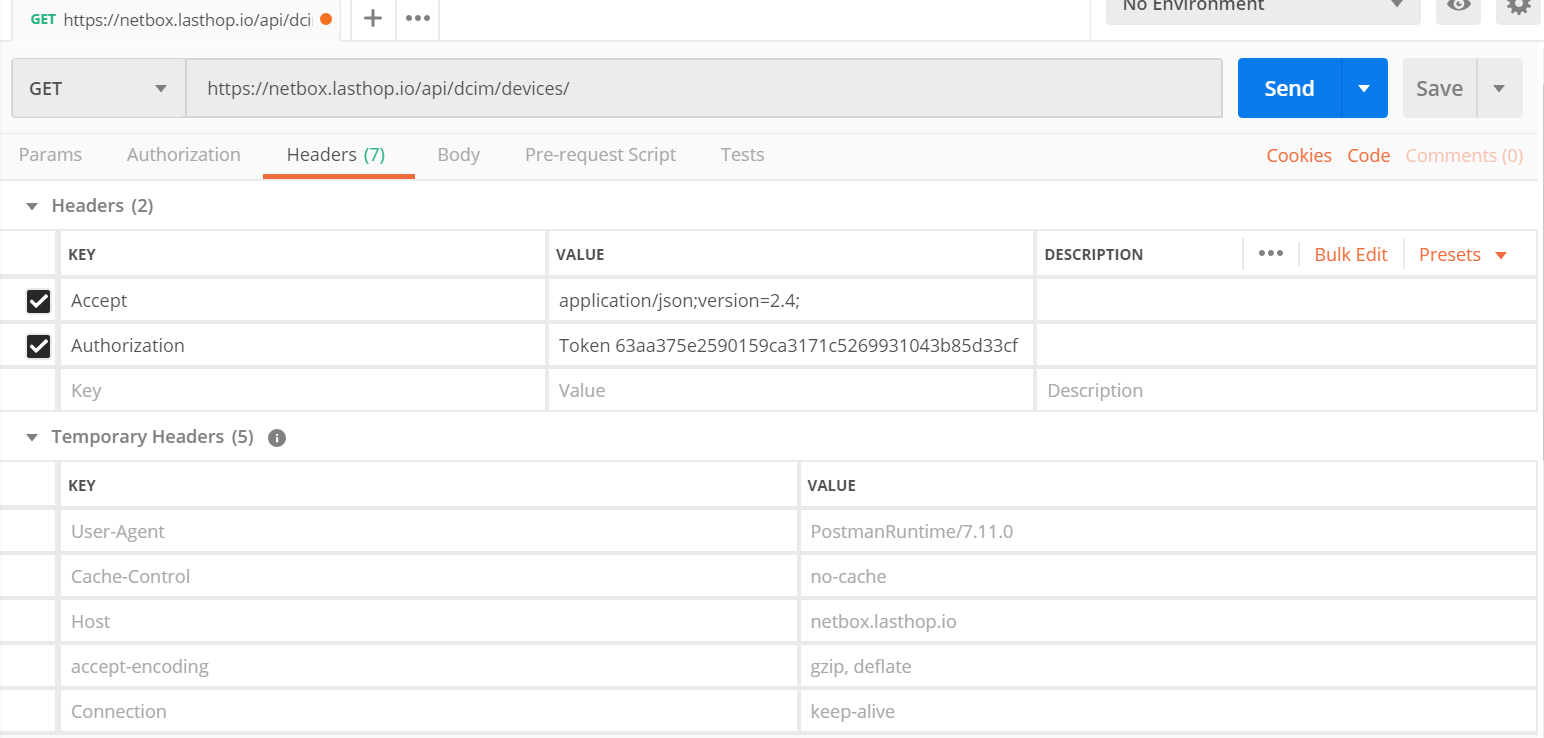
19:31

Since REST API’s are so common and they're so readily available, they also have a lot of tools that might be available to use.

I'm showing you, here, one tool that's very common to use with REST API’s and this is something known as Postman. You can see here that I have my URL "<https://netbox.lasthop.io/api/> that I'm accessing.

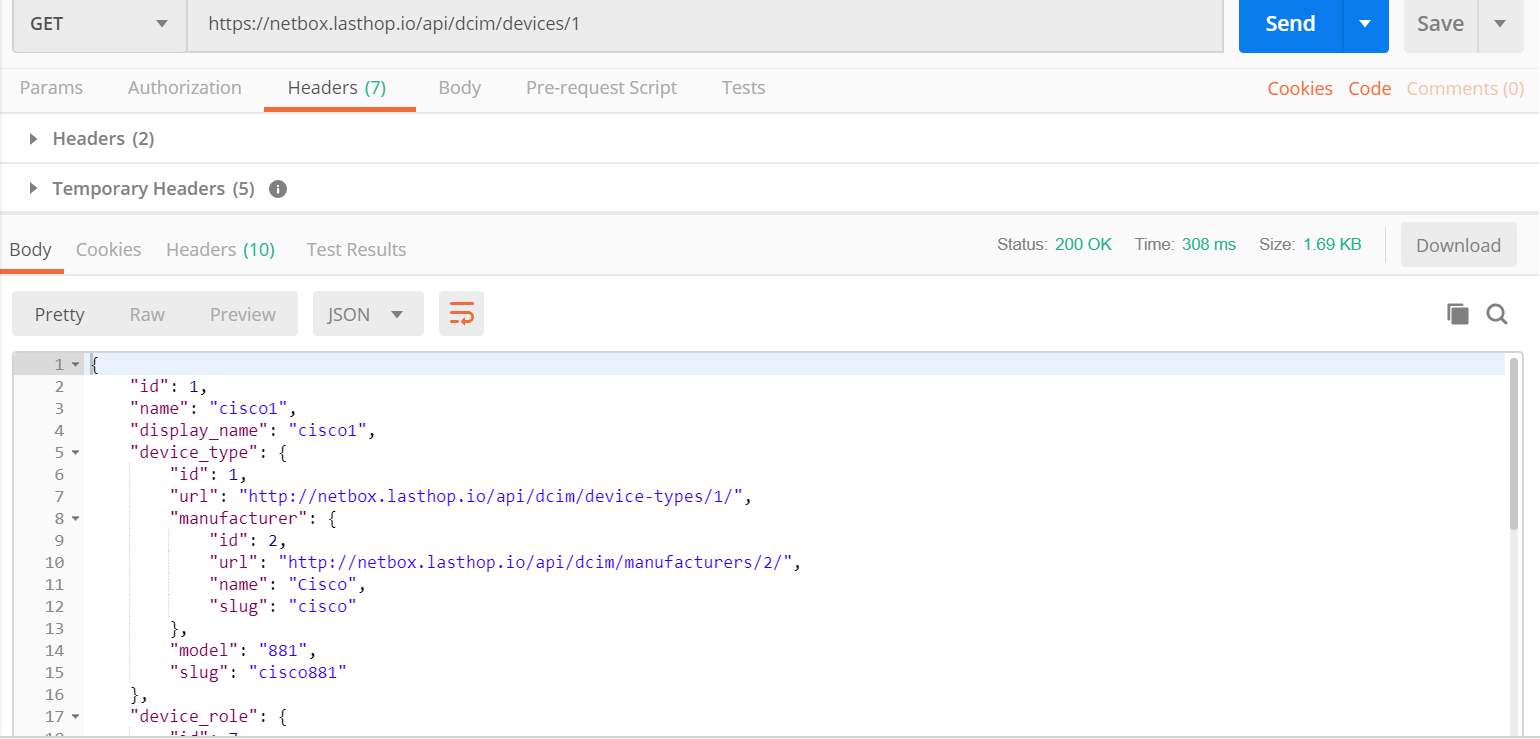


Here I'm accessing the ‘<https://netbox.lasthop.io/api/dcim/devices>’.

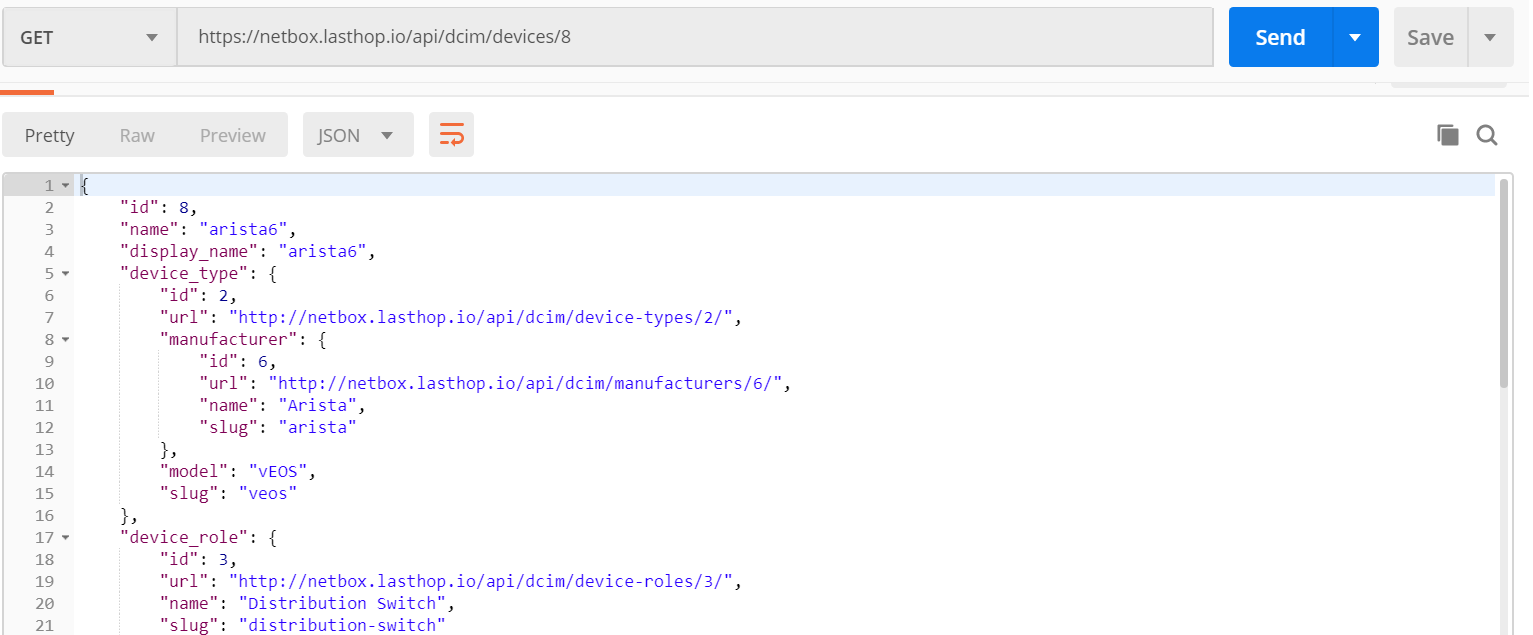


Now notice this actually requires the token, so I've added this accept. I said, ‘application/json;version=2.4;’. I sort of worked this out from looking at the API documentation, and from lookingat this PyNetBox library, seeing how the GET should be constructed when it required authorization.Then I added this authorization field. Now notice that these are both HTTP headers. Here's my token, which is just a string here, then I have a space, then I have my actual token. So in theHTTP headers I'm adding this, ‘application/json; version=2.4;’, and I'm also adding this authorization, and then this token string, and then the actual token.

If go ahead and put the device id "1", it gives me the first device there, which is this ‘Cisco1'.



If I do device ID 8, it gives me the arista6.



Pyplus - REST API GET using Python

jeudi, 2 mai 2019

12:20

Let's start interfacing to a REST API using Python. The first thing we're going to do when we're interfacing to a REST API is you're almost always going to want to use the Python ‘requests’ library. This is the standard, basically the default way of interfacing to REST API’s in Python. It is ‘pip’ installable so it's not included in the standard library, so you have to install it using ‘pip’.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 ~]$ pip install requests

Collecting requests

Using cached <https://files.pythonhosted.org/packages/7d/e3/20f3d364d6c8e5d2353c72a67778eb189176f08e873c9900e10c0287b84b/requests-2.21.0-py2.py3-none-any.whl>

Requirement already satisfied: certifi>=2017.4.17 in ./VENV/py3\_venv/lib/python3.6/site-packages (from requests) (2018.11.29)

Requirement already satisfied: idna<2.9,>=2.5 in ./VENV/py3\_venv/lib/python3.6/site-packages (from requests) (2.8)

Requirement already satisfied: urllib3<1.25,>=1.21.1 in ./VENV/py3\_venv/lib/python3.6/site-packages (from requests) (1.24.1)

Requirement already satisfied: chardet<3.1.0,>=3.0.2 in ./VENV/py3\_venv/lib/python3.6/site-packages (from requests) (3.0.4)

Installing collected packages: requests

Successfully installed requests-2.21.0

I have to do import ‘requests’, I import pretty print [‘pprint’].

import requests

from pprint import pprint

This "from urllib3.exceptions import InsecureRequestWarning" is just to make the SSL exception, the SSLcertificate warning go away. Basically I'm using a device that has an unsigned SSL certificate, so there's going to be a warning from that, and I'm just saying disable that warning for now.

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

Now we're in the main part of our program. We're going to specify a URL of NetBox. Now all we're doing here is, we're doing an HTTP GET. Notice I specified the ‘http\_headers’. When I actually do my call to the request library, I specify ‘request.get’. I specify the ‘url’, specify the ‘headers’. I use the ‘http\_headers’ variable that I created here.

if \_\_name\_\_ == "\_\_main\_\_":

url = "<https://netbox.lasthop.io/api/>"

http\_headers = {"accept": "application/json; version=2.4;"}

response = requests.get(url, headers=http\_headers, verify=False)

response = response.json()

I started debugging and looking at what the request format looked like from this PyNetBox library. You might have also been able to glean this potentially from some of our other tools that we were looking at, but basically I started doing debugging on the PyNetBox and looking at the nature of a GET request when it was constructed by that library. The ‘verify=False’ is just saying, "don't check the SSL. Don't worry about the SSL certificate." This is the request library, so it's saying the SSL certificate not being validated is okay from the request library perspective. I get the response back. This ‘response’ object, I actually get the JSON payload from that. That automatically converts this over to a Python data structure, and then I pretty print that out.

Here is the python code to pull the information

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat get\_request.py

import requests

from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

url = "<https://netbox.lasthop.io/api>"

http\_headers = {"accept": "application/json; version=2.4;"}

response = requests.get(url, headers=http\_headers, verify=False)

response = response.json()

print()

pprint(response)

print()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ python get\_request.py

{'circuits': '<http://netbox.lasthop.io/api/circuits/>',

'dcim': '<http://netbox.lasthop.io/api/dcim/>',

'extras': '<http://netbox.lasthop.io/api/extras/>',

'ipam': '<http://netbox.lasthop.io/api/ipam/>',

'secrets': '<http://netbox.lasthop.io/api/secrets/>',

'tenancy': '<http://netbox.lasthop.io/api/tenancy/>',

'virtualization': '<http://netbox.lasthop.io/api/virtualization/>'}

Pyplus - REST API GET with Authentication using Python

jeudi, 2 mai 2019

15:43

I am going to access device ID 1 in the devices page using the same python code but the URI is changed to access the devices page and let see what happens. You can see it needs Authentication.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat get\_request.py

import requests

from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

url = "<https://netbox.lasthop.io/api/dcim/devices/>"

http\_headers = {"accept": "application/json; version=2.4;"}

response = requests.get(url, headers=http\_headers, verify=False)

response = response.json()

print()

pprint(response)

print()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ python get\_request.py

{'detail': 'Authentication credentials were not provided.'}

Let us now add token to the headers for the authentication to work and see what happens. There are 2 ways to add the tokens.

Here is first method,

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat get\_request\_auth.py

import requests

from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

url = "<https://netbox.lasthop.io/api/dcim/devices/1>"

http\_headers = {"accept": "application/json; version=2.4;", "authorization": "Token 63aa375e2590159ca3171c5269931043b85d33cf"}

response = requests.get(url, headers=http\_headers, verify=False)

response = response.json()

print()

pprint(response)

print()

Here is the second method,

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat get\_request\_auth.py

import requests

from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

token = "63aa375e2590159ca3171c5269931043b85d33cf"

url = "<https://netbox.lasthop.io/api/dcim/devices/1>"

http\_headers = {"accept": "application/json; version=2.4;"}

if token:

http\_headers["authorization"] = "Token {}".format(token)

response = requests.get(url, headers=http\_headers, verify=False)

response = response.json()

print()

pprint(response)

print()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ python get\_request\_auth.py

{'asset\_tag': None,

'cluster': None,

'comments': '',

'config\_context': {},

'created': '2018-10-25',

'custom\_fields': {},

'device\_role': {'id': 7,

'name': 'Router',

'slug': 'router',

'url': '<http://netbox.lasthop.io/api/dcim/device-roles/7/>'},

'device\_type': {'id': 1,

'manufacturer': {'id': 2,

'name': 'Cisco',

'slug': 'cisco',

'url': '<http://netbox.lasthop.io/api/dcim/manufacturers/2/>'},

'model': '881',

'slug': 'cisco881',

'url': '<http://netbox.lasthop.io/api/dcim/device-types/1/>'},

'display\_name': 'cisco1',

'face': {'label': 'Front', 'value': 0},

'id': 1,

'last\_updated': '2018-10-26T15:09:15.249257-07:00',

'local\_context\_data': None,

'name': 'cisco1',

'parent\_device': None,

'platform': {'id': 1,

'name': 'Cisco IOS',

'slug': 'cisco-ios',

'url': '<http://netbox.lasthop.io/api/dcim/platforms/1/>'}

Pyplus - REST API Python Create Object

jeudi, 2 mai 2019

17:19

we're going to try to create a device on the NetBox. On our NetBoxsystem, we're going to try to create a device. We have our glue code with some imports. Here'sour ‘requests’. We import the ‘json’ library,

I wanted to get information about what the format of [what] our HTTP POST needed to look like. When we'recreating an object, we're going to use a HTTP POST. We're going to specify the resource. We'regoing to POST to, and it's going to be this resource here, which is the ‘NetBox DCIM devices’. We specify our authorization in the exact same way we specified it before. Then we're going to pass in some ‘POST DATA’. This data that's going to specify what is it thatwe are creating and the following is the format.

"name": "cisco3",

"display\_name": "cisco3",

"device\_type": 1 # 881

"device\_role": 7, # Router

"platform": 1, # Cisco IOS

"rack": 1, # RK1

"site": 1, # Fremont Data Center

"status": 1, # Active

Now, here's my ‘requests’ library, and then I say, ‘.post’ on that [‘requests.post(‘]. I'm going toPOST to the ‘url’ which was that device’s URL. I'm going to pass in the ‘http\_headers’. Nownotice here, I have an additional field which is ‘data’, that I am going to pass in on my POST.The data we're passing in is actually going to be inside that HTTP, and it's actually going topassed in as JSON data.Here, I actually call my JSON library. I say ‘json.dumps’, so it is going to dump it as string and Ipass in this ‘post\_data’, data structure [data=json.dumps(post\_data)]. This is going to beconverted to a JSON string, and then sent down as part of this ‘request.post’. I turn off the SSL verification.

Now the complete code look like this

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat post\_request.py

import requests

import json

from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

token = "63aa375e2590159ca3171c5269931043b85d33cf"

url = "<https://netbox.lasthop.io/api/dcim/devices/>"

http\_headers = {

"Content-Type": "application/json; version=2.4;",

"Authorization": "Token {}".format(token)

}

post\_data = {

"name": "cisco3",

"display\_name": "cisco3",

"device\_type": 1, # 881

"device\_role": 7, # Router

"platform": 1, # Cisco IOS

"rack": 1, # RK1

"site": 1, # Fremont Data Center

"status": 1 # Active

}

response = requests.post(url, headers=http\_headers, data=json.dumps(post\_data), verify=False)

response = response.json()

print()

pprint(response)

print()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ python post\_request.py

{'asset\_tag': None,

'cluster': None,

'comments': '',

'created': '2019-05-02',

'device\_role': {'id': 7,

'name': 'Router',

'slug': 'router',

'url': '<http://netbox.lasthop.io/api/dcim/device-roles/7/>'},

'device\_type': {'id': 1,

'manufacturer': {'id': 2,

'name': 'Cisco',

'slug': 'cisco',

'url': '<http://netbox.lasthop.io/api/dcim/manufacturers/2/>'},

'model': '881',

'slug': 'cisco881',

'url': '<http://netbox.lasthop.io/api/dcim/device-types/1/>'},

'display\_name': 'cisco3',

'face': None,

'id': 16,

'last\_updated': '2019-05-02T08:11:29.412432-07:00',

'local\_context\_data': None,

'name': 'cisco3',

'parent\_device': None,

'platform': {'id': 1,

'name': 'Cisco IOS',

'slug': 'cisco-ios',

'url': '<http://netbox.lasthop.io/api/dcim/platforms/1/>'},

'position': None,

'primary\_ip': None,

'primary\_ip4': None,

'primary\_ip6': None,

'rack': {'display\_name': 'RK1',

'id': 1,

'name': 'RK1',

'url': '<http://netbox.lasthop.io/api/dcim/racks/1/>'},

'serial': '',

'site': {'id': 1,

'name': 'Fremont Data Center',

'slug': 'fremont-dc',

'url': '<http://netbox.lasthop.io/api/dcim/sites/1/>'},

'status': {'label': 'Active', 'value': 1},

'tags': [],

'tenant': None,

'vc\_position': None,

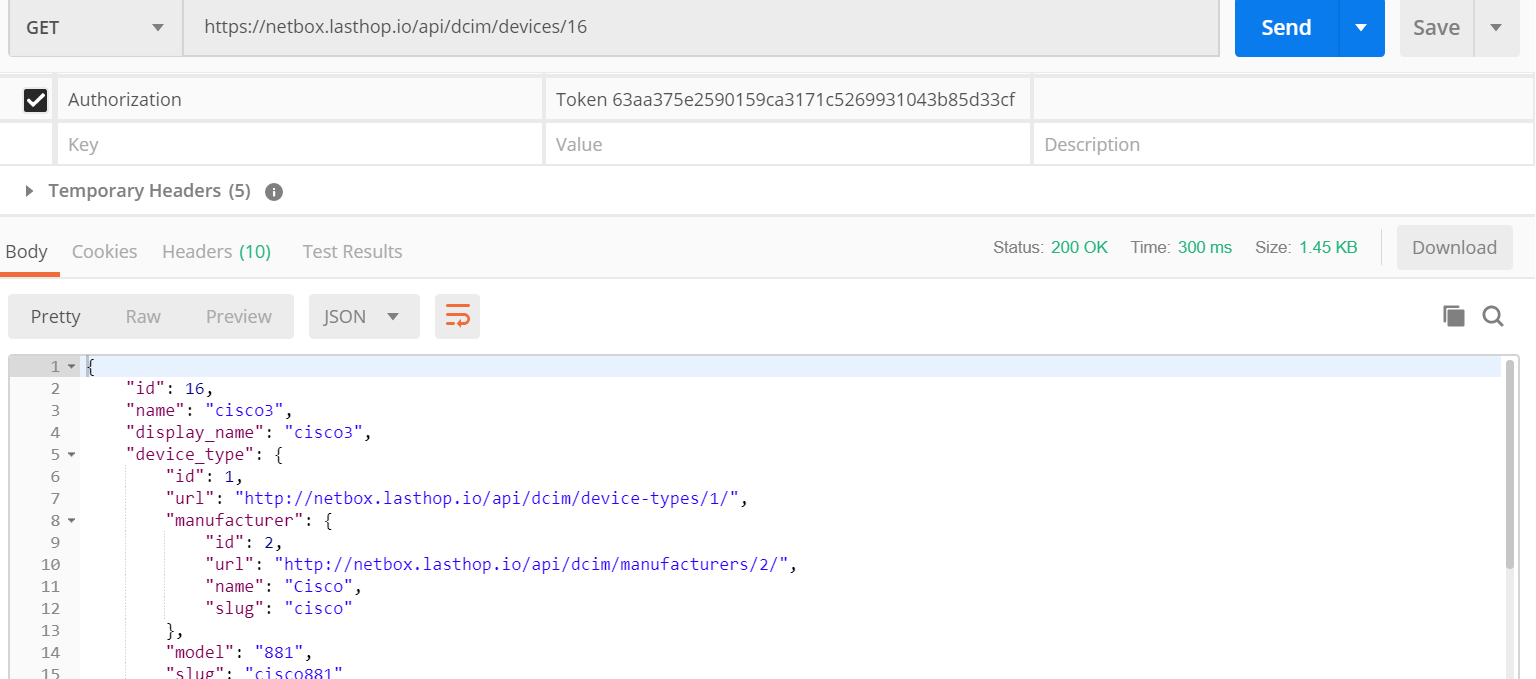
'vc\_priority': None,

'virtual\_chassis': None}

I checked if the device has been created under device page using the postman GET to <http://netbox.lasthop.io/api/dcim/devices> and I could see this

"created": "2019-05-02",

"last\_updated": "2019-05-02T08:11:29.412432-07:00"



Pyplus - REST API Python Modify Object

jeudi, 2 mai 2019

17:32

Let's now look at how we would modify an existing object using the Python Requests library.

Now, the first thing I do is, I actually retrieve the current object. I go ahead and get the cisco3 object that I created in earlier example. We were doing the ‘GET’, we have this as ‘accept’. When we're doing the HTTP PUT, we switched that to ‘ContentType’. So we update our ‘http\_headers.

Now, one thing I had to do with the PUT, is that I have to actually modify the existing object. If we look at this device, this Arista6, we can see that we have fields like ‘device\_type’ and

‘device\_role’. But if we look at what comes back for ‘device\_type’, we actually get a dictionary back. That dictionary contains a set of fields, like ‘id’, the ‘url’ that corresponds to how do we retrieve that, a model associated with that, and more details. This actually links to a manufacturer in that case, which has another id.

But, when we're doing the PUT operation, all it wants is the ‘id’. It doesn't want all these

additional references. It'll actually give you an error if you try to pass this entire dictionary for the ‘device\_type’. It'll give me an error if I try to pass this entire ‘device\_role’ here, as opposed to just passing the ‘id’ of 16.

Set the status field to ‘1’. The status field indicates the device is active. I'm going to change this cisco3 device to be in Rack2. It's actually on Rack1. Right now, it references RK1. Let's modify to say that, that belongs to RK2. Now, this is an ‘id’. Once again, we'd have to look up that this number 2 actually maps to the rack that we want it to be in.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat put\_request.py

import requests

import json

from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

token = "63aa375e2590159ca3171c5269931043b85d33cf"

url = "<https://netbox.lasthop.io/api/dcim/devices/16/>"

http\_headers = {

"accept": "application/json; version=2.4;",

"authorization": "Token {}".format(token)

}

# Retrieve the information as a first step

response = requests.get(url, headers=http\_headers, verify=False)

cisco3 = response.json()

http\_headers = {

"Content-Type": "application/json; version=2.4;",

"authorization": "Token {}".format(token)

}

# Reformat to get the proper structure for the existing object

for field in ["device\_role", "device\_type", "platform", "site", "rack"]:

cisco3[field] = cisco3[field]["id"]

cisco3["status"] = 1

cisco3["rack"] = 2

modify\_device = requests.put(

url, headers=http\_headers, data=json.dumps(cisco3), verify=False

)

response = modify\_device.json()

print()

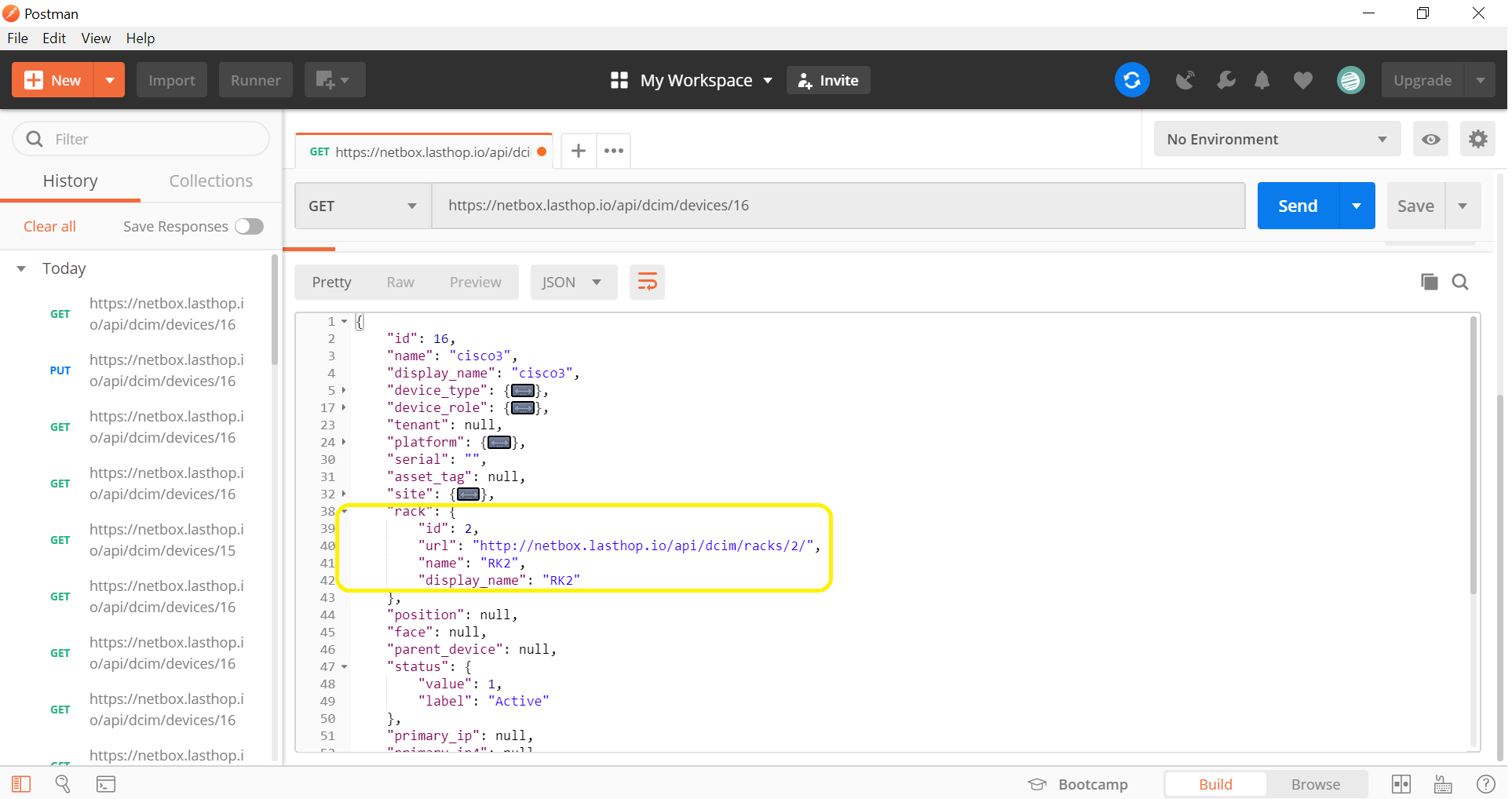
pprint(response)

print()

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ python put\_request.py | grep rack

'rack': {'display\_name': 'RK2',

'url': '<http://netbox.lasthop.io/api/dcim/racks/2/>'}



Pyplus - REST API Python Delete Object

vendredi, 3 mai 2019

10:20

In this I am going to delete the object that I created. The device that I created has the ID 17.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat delete\_request.py

import requests

import json

from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

token = "63aa375e2590159ca3171c5269931043b85d33cf"

url = "<https://netbox.lasthop.io/api/dcim/devices/17/>"

http\_headers = {

"Content-Type": "application/json; version=2.4;",

"Authorization": "Token {}".format(token)

}

response = requests.delete(url, headers=http\_headers, verify=False)

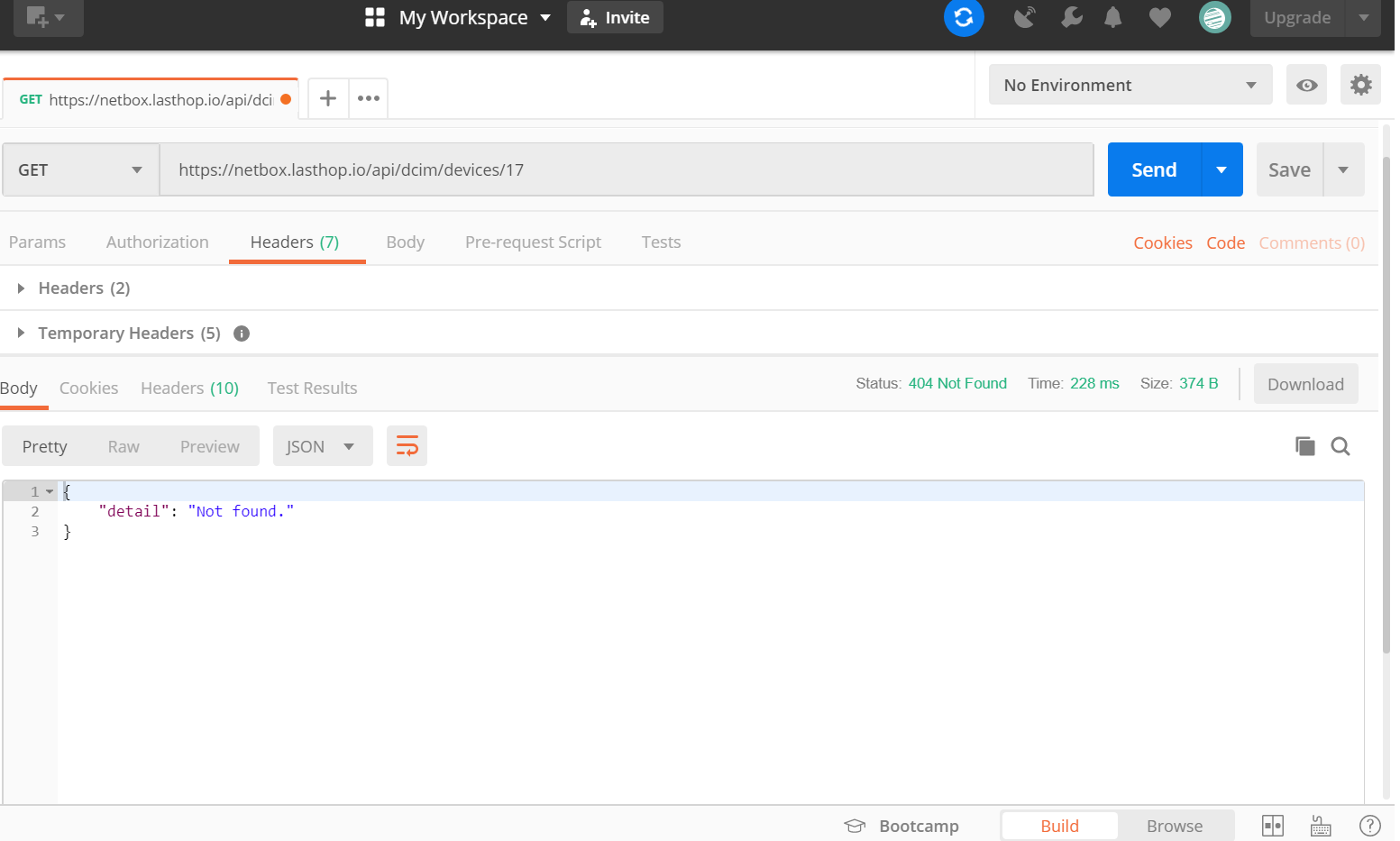
if response.ok:

print("Device cisco3 deleted")

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ python delete\_request.py

Device cisco3 deleted

I can see the device deleted from the python output, but I will check it using the postman GET by accessing the device page with device with ID 17.



Pyplus - REST API Authentication

vendredi, 3 mai 2019

11:05

REST API authentication in a more general manner, or in a more general way, and I want to show you some patterns that I've seen across time for doing authentication to REST API’s.

A lot of the times, dealing with a REST API, authentication iswhere you can sort of run into issues as far as how to make the authentication work. We've looked at, previously, the NetBox authorization format. The NetBox authorization formatis we add this ‘authorization’ into the ‘http\_headers’ and we get a ‘token’ and that ‘token’ is something that we've retrieved from NetBox.

Now if we look at the GitHub authentication, or one way of doing the GitHub authentication, it's basically this standard basic authentication. With GitHub, we import this ‘HTTPBasicAuth’ fromthe ‘request.auth’ library. Then we can do the following. We can do ‘requests.get’, we specifyour URL, we specify our ‘http\_headers’. Notice the ‘http\_headers’ don't have anything at thispoint on authorization. We then say ‘auth=HTTPBasicAuth’. We specify our username andpassword. This pattern is so common that there's a short and way of doing it in the request library.You can also just say ‘auth=(username, password)’. What happens under the hood is that itactually adds an HTTP authorization header using the basic format and then the basic, there is an encrypted format, well not encrypted, there is a way the username and password are encoded in Base64 encoding.

It's very trivial to decode that. You would definitely want to use this in conjunction with some sortof encryption like your standard TLS encryption because it's very trivial to decode that. I didsome experimentation with decoding that Base64 encoding, and it basically takes no effort to dothat, or very little effort to do that. This is basic authentication using the requests library. This iswhat we're doing in this form of GitHub. GitHub also supports some OAuth forms ofauthentication as well

Now, let's look at some other networking related platforms or devices.

Here's Infoblox, you can also do ‘basic Auth’ with Infoblox. I got this literally from some Infobloxlibrary, I modify it slightly because they were using it in a class, so I remove some of this ‘self.’references. Here's our URL, here's our ‘auth=’ username and password [‘auth=(iba\_user,iba\_password)’]. Then here's our "Do we want to verify the SSL certificate or not?" So, a similarpattern that we have seen before.

Now, here we look at ‘NSoT’. ‘NSoT’ is Networks Source of Truth, it is also another sort of IPAM,DCIM management platform. This came out of Dropbox. So Dropbox, this ‘NSoT’ is more similarto what we had seen with NetBox. We basically get a token back. We're going to get a tokenback and we actually get the token by querying the device. So there's a certain way that youquery the API and remote management server to get the token back.Then you use this token in your HTTP authorization header, and it has this format of‘AuthToken’, email, and the token.There's certain advantages with using the token, in that if your username and password arecompromised, the person that has retrieved those could do other malicious things that could bebad, like change your username and password and lock you out of that account depending onwhat it is. Whereas, if it's the token, then at least you still have your primary username andpassword credentials where you can go in, change the token, or declare the previous token asbeing invalid.

Now let's look at the Cisco ASA REST API, it is also using a BasicAuth. This is from a library,that was a Python library that was interfacing to the Cisco ASA REST API. You can see onceagain that we're just doing a basic authentication here. We have our ‘auth=self.cred’, and whatwas ‘self.cred’? ‘self.cred’ was just this ‘HTTPBasicAuth’ with username and password. So justlike we had seen previously with the GitHub basic authentication, we're using that here with theCisco REST API.One last example, and this is with the Cisco APIC. This one's a little bit more interesting. Withthe Cisco APIC, you actually have this ASA login, or this ‘aaaLogin.json’, URL. You actuallyhave to create this JSON object, and there's a way you can do this with XML. It has this veryparticular format, so you have to have this structured in a certain way. Ultimately, you have yourusername and password embedded in here.

Now the Cisco APIC actually uses a cookie. The easiest way to do this in the Requests library isuse this ‘request.Session()’ to create a session object. Your session object will automatically store your cookie, and use your cookie after you have it, going forward. So once you do this,then you can create this ‘my\_session.post’, you post to this URL, we're posting to this‘aaaLogin’. We pass in our ‘json\_creds’, and then we go ahead and that contains that data. Thiswill get causes to get this APIC cookie back, that'll be stored in our session, and then we canuse that later on.I went and engineered this based on what I saw for the APIC documentation. Then I happenedto have somebody that was testing, using APIC and they said that it worked properly for them.So these are some patterns that you see with REST API authentication.

Pyplus - REST API - How to Begin

vendredi, 3 mai 2019

15:43

In general, when you start using a new REST API, the very first thing you want to do is you wantto find out what API documentation it has available to you. You want to start constructing verysimple use cases for how to use that REST API. In particular, you want to start with, “How do Iretrieve information from this REST API?”, and start with the most simple case you can. Then tryto layer things on top of that, and basically always have working code in that process.In addition to looking for examples of how to use the REST API, you also want to do a searchand find out is there any Python library available for this particular API. In a lot of cases, therewill be an already existing Python library available for you. In general, you want to use a librarythat's available for you to help you simplify the mechanics of dealing with the REST API.The advantage of using an existing library is, a lot of the low level mechanics of how to constructthe HTTP URL's, and how to construct the HTTP headers, and how to handle theauthentication, those processes will already have been solved for you.

You can, in a much more Pythonic way, deal with the things that you want to accomplish usingthat existing library.If that's the case, you can rightfully ask me, "Why the heck are we even talking about all theselow level mechanics?" The reason we're talking about them is one, whenever anything goeswrong, it is very helpful to have a framework for what is happening here under the hood. Whymight my library be going wrong? How do I get the needed information for me to solve theproblems that I'm encountering using the API? You're almost always going to be having someset of problems using the API, and you need a set of tools to help you work through thoseproblems.

The second reason is you're going to potentially run into some cases where there's no goodPython library available to you. Or, there might be a Python library, but it doesn't do everythingyou need it to do. Or it has some undesirable behaviors, undesirable problems that you're goingto need to work through, and potentially modify that library. You need enough of anunderstanding of what's going on, so that you can deal with those problems, and handle theissues that arise

Pyplus - Python Code Style

vendredi, 3 mai 2019

16:06

At a high level, here I'm pointing to the PEP 8 document, you can see it's just standard. It's pretty straightforward. It starts talking about a lot of things like:

Indentation

Should you use tabs or spaces?

What's the maximum line length?

How should we use white space in different contexts?

How we should do commenting

Certain naming conventions

It's a whole bunch of stylistic conventions

Now, at a high level, some of the things that I do think matter more than other things. I do thinkline lengths matter. If you get too long of line lengths, for example, and you can talk about wherethis starts to kick in. If you get up to 140 character line lengths or 150 character line lengths,you're going to start getting problems. Those problems are, you're going to get things like linewrapping. So you post these up into Git, up into GitHub, your code is going to line wrap. Or ifsomebody is using smaller terminal sizes, you're going to start getting line wrapping, which isgoing to make your code much harder to read. Now, you can talk about where is the appropriate boundary for line lengths. I think 100characters is a reasonable compromise between being too short and too long, and you can talkabout where exactly that should be.

Another set of things that I think matter are naming conventions. In general, Python variablenames are all lowercase, and they have an underscore for word separators. Similarly, forfunctions and for Python methods, once again, they are all lowercase. They use underscores forword separators. Class names use Camel Case. Camel Case would be as follows, uppercasefor the first letter, uppercase for new words, and then there are no spaces, no underscores inthe class names. For Constants, you would use all uppercase with underscores as wordseparators.I do find those to be helpful because it makes it easier when you look at something in Pythoncode to know what it is. “Oh, this is probably a class.” “Oh, this is probably a variable.” “Oh, thisis probably a function name, a method name.” So it does make it easier when you see thosethings in the code to have an inference as to what it is.

About python tools, for how to enforce linting, to how tocheck linting these are things like pylint, pycodestyle. We're also going to talk about a tool named Black that can help you autoformat your code correctly.

Pyplus - Pylint

samedi, 4 mai 2019

10:19

If you're a new to Python, you're not going to know all ofthese coding conventions and coding standards. So you want tools that at least flag this for you,and even better would be, they autoformat for you. Let's start looking at some of these toolsthat are going to enable this for us. Really in Python, there are two main tools that will do this forus.The first one is called Pylint and the second one is called pycodestyle. Let's look at someexamples of using each of these tools, we'll start out with Pylint. So you'll see that I have Pylintinstalled, you can see if I do ‘pip list’ and I ‘grep’ for ‘pylint’, you're going to see it there.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 /]$ pip install pylint

Successfully installed astroid-2.2.5 pylint-2.3.1 typed-ast-1.3.5

(py3\_venv) [sethuvignesh@ip-172-30-0-144 /]$ pip list | grep pylint

pylint 2.3.1

I am executing the code with pylint to see if my code matches with the PEP8 format. You can see there are many linters highlighted and at the end I have received negative score for my program coding style. Though code executes properly it is not matching the proper python coding format.

Here is my code and I am going to execute it with Pylint

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat delete\_request.py

import requests

import json

from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

token = "63aa375e2590666ca3171c5269931043b85d33cf"

url = "<https://netbox.lasthop.io/api/dcim/devices/17/>"

http\_headers = {

"Content-Type": "application/json; version=2.4;",

"Authorization": "Token {}".format(token)

}

response = requests.delete(url, headers=http\_headers, verify=False)

if response.ok:

print("Device cisco3 deleted")

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ pylint delete\_request.py

\*\*\*\*\*\*\*\*\*\*\*\*\* Module delete\_request

delete\_request.py:11:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)

delete\_request.py:12:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)

delete\_request.py:13:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)

delete\_request.py:14:0: C0330: Wrong hanging indentation (remove 2 spaces).

"Content-Type": "application/json; version=2.4;",

| ^ (bad-continuation)

delete\_request.py:15:0: C0330: Wrong hanging indentation (remove 2 spaces).

"Authorization": "Token {}".format(token)

| ^ (bad-continuation)

delete\_request.py:16:0: C0330: Wrong hanging indentation.

}

| |^ (bad-continuation)

delete\_request.py:18:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)

delete\_request.py:19:0: C0303: Trailing whitespace (trailing-whitespace)

delete\_request.py:20:0: W0311: Bad indentation. Found 2 spaces, expected 4 (bad-indentation)

delete\_request.py:21:0: W0311: Bad indentation. Found 4 spaces, expected 8 (bad-indentation)

delete\_request.py:1:0: C0111: Missing module docstring (missing-docstring)

delete\_request.py:7:0: E1101: Module 'requests.packages' has no 'urllib3' member (no-member)

delete\_request.py:11:2: C0103: Constant name "token" doesn't conform to UPPER\_CASE naming style (invalid-name)

delete\_request.py:12:2: C0103: Constant name "url" doesn't conform to UPPER\_CASE naming style (invalid-name)

delete\_request.py:13:2: C0103: Constant name "http\_headers" doesn't conform to UPPER\_CASE naming style (invalid-name)

delete\_request.py:18:2: C0103: Constant name "response" doesn't conform to UPPER\_CASE naming style (invalid-name)

delete\_request.py:2:0: W0611: Unused import json (unused-import)

delete\_request.py:3:0: W0611: Unused pprint imported from pprint (unused-import)

delete\_request.py:2:0: C0411: standard import "import json" should be placed before "import requests" (wrong-import-order)

delete\_request.py:3:0: C0411: standard import "from pprint import pprint" should be placed before "import requests" (wrong-import-order)

-------------------------------------

Your code has been rated at -10.00/10

I am going to correct the linters see if my score is increasing. You can see after all the modification I made the score is 10/10.

Here is my modified code and score.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ cat delete\_request.py

""" This is a formatted code as per PEP8 Style format"""

#import json

from pprint import pprint

import requests

#from urllib3.exceptions import InsecureRequestWarning

#requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

TOKEN = "63aa375e2590666ca3171c5269931043b85d33cf"

URL = "<https://netbox.lasthop.io/api/dcim/devices/17/>"

HTTP\_HEADERS = {

"Content-Type": "application/json; version=2.4;",

"Authorization": "Token {}".format(TOKEN)

}

RESPONSE = requests.delete(URL, headers=HTTP\_HEADERS, verify=False)

if RESPONSE.ok:

pprint("Device cisco3 deleted")

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-RESTAPI]$ pylint delete\_request.py

-------------------------------------------------------------------

Your code has been rated at 10.00/10 (previous run: 8.89/10, +1.11)

Pyplus - Pycode style

lundi, 13 mai 2019

14:02

Pylint worries about too much minutiae, and I think pycodestyle is a bit better than that. we run pycodestyle, same exact way as we saw with Pylint. Here is my sample program and I am going to run this program against pycodestyle.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat pycodestyle\_test.py

import json

from pprint import pprint

import requests

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

token = "63aa375e2590159ca3171c5269931043b85d33cf"

url= "<https://netbox.lasthop.io/api/dcim/devices/18/>"

http\_headers = {

"Content-Type": "application/json; version=2.4;",

"Authorization": "Token {}".format(token)

}

response = requests.delete(url, headers=http\_headers, verify=False)

if response.ok:

pprint("Device cisco3 deleted")

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pycodestyle pycodestyle\_test.py

pycodestyle\_test.py:9:3: E111 indentation is not a multiple of four

pycodestyle\_test.py:10:3: E111 indentation is not a multiple of four

pycodestyle\_test.py:10:6: E225 missing whitespace around operator

pycodestyle\_test.py:11:3: E111 indentation is not a multiple of four

pycodestyle\_test.py:15:5: E113 unexpected indentation

pycodestyle\_test.py:17:8: E111 indentation is not a multiple of four

I will correct the errors and will see if how good is my code formatting

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pycodestyle pycodestyle\_test.py

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$

You can see there are no errors.

Here is my code after formatting

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat pycodestyle\_test.py

import json

from pprint import pprint

import requests

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

token = "63aa375e2590159ca3171c5269931043b85d33cf"

url = "<https://netbox.lasthop.io/api/dcim/devices/18/>"

http\_headers = {

"Content-Type": "application/json; version=2.4;",

"Authorization": "Token {}".format(token)

}

response = requests.delete(url, headers=http\_headers, verify=False)

if response.ok:

pprint("Device cisco3 deleted")

Pyplus - Pylama

lundi, 13 mai 2019

14:19

I'm going to talk about a toolcalled Pylama. What Pylama does is it actually wraps multiple tools. Pylama will invoke, for you,tools like pycodestyle, or tools like Pylint. There's additional tools that it can also invoke. I typically use Pylama with pycodestyle and Pyflakes. Pyflakes is more of checking for errors or problems in your code, pycodestyles is a linter that we've already seen.If you wanted to use Pylint instead, you could swap in Pylint. But you can see there's a set ofutilities that you can call and invoke using Pylama. Once again, Pylama is installed using pip. So we can say ‘which pylama’, we can say ‘pip list | grep pylama’. We can see that we've installedat using pip.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 netmiko\_tools]$ which pylama

~/VENV/py3\_venv/bin/pylama

(py3\_venv) [sethuvignesh@ip-172-30-0-144 netmiko\_tools]$ pip list | grep pylama

pylama 7.6.6

I will almost always create a ‘setup.cfg’ file for it. Sothis specifies my configuration for Pylama. This actually should be the PyCode, the ‘setup.cfg’from the Netmiko repository.What I would probably do as a starting point is, I would actually take this setup that got‘setup.cfg’ from Netmiko. I would use it as a starting point, and then I would edit it as needed.You can see here that I'm invoking three different utilities from Pylama. I'm invoking somethingcalled ‘mccabe,pep8’ ([PEP 8] which is another name for pycodestyle). That's what pycodestyleused to be known as, and then the Pyflakes utility. You can see there's certain rules that I set, Idon't want you to care about these rules, you can completely ignore these rules.I'd have to actually go look up what those were, but I decided that those weren't valuable. Thosesets of checks weren't valuable, so I didn’t turn them on. There's a set of directories that you tellPylama to skip

(py3\_venv) [sethuvignesh@ip-172-30-0-144 netmiko\_tools]$ cat setup.cfg

[pylama]

linters = mccabe,pep8,pyflakes

ignore = D203,C901

[pylama:pep8]

max\_line\_length = 100

(py3\_venv) [sethuvignesh@ip-172-30-0-144 netmiko\_tools]$ pwd

/home/sethuvignesh/netmiko\_tools

Now notice here I've actually specified a default for PEP 8, I.e., pycodestyle, that I want to beoverwritten, and that is this max line length. I said, “Instead of using the default max line length, Iwant you to use a hundred characters as the max line length”, which I previously mentioned thatI thought was a reasonable compromise between lines too long and lines too short. Now just like our other utilities, we can do ‘pylama .’, and that will check the current directoryand all its subdirectories, all the Python files

Here is my sample code and let me run the code using pylama.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat pylama\_test.py

import json

from pprint import pprint

import requests

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

if \_\_name\_\_ == "\_\_main\_\_":

token = "63aa375e2590159ca3171c5269931043b85d33cf"

url = "<https://netbox.lasthop.io/api/dcim/devices/18/>"

http\_headers = {

"Content-Type": "application/json; version=2.4;",

"Authorization": "Token {}".format(token)

}

response = requests.delete(url, headers=http\_headers, verify=False)

if response.ok:

pprint("Device cisco3 deleted")

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pylama pylama\_test.py

WARNING:root:Linter `eradicate` not found.

pylama\_test.py:1:1: W0611 'json' imported but unused [pyflakes]

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pip install eradicate

Collecting eradicate

Installing collected packages: eradicate

Successfully installed eradicate-1.0.

Let me comment out the line "import json" and see if it negates the error highlighted by pyflakes.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat pylama\_test.py

# import json

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pylama pylama\_test.py

pylama\_test.py:1:1: E800: Found commented out code: # import json [eradicate]

Now the eradicate highlights the commented out line in the code. So I am removing the commented out line from the code and run the code with Pylama. You can see the code is clean.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pylama pylama\_test.py

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$

Pyplus - Black

lundi, 13 mai 2019

16:34

Python Black is an auto formatter of your code. One of the problems you can get into when you're working with linters is that the actual code formatting itself of fixing all the issues with the linter flags can be very, very time consuming. It can suck up a lot of time.A second issue that you can run into is, you can just get into these pointless arguments between various people of what the correct formatting should be. You'll have one person that'll want it to be styled in format one, and person two that'll want it to be styled in format two. It can just take alot of time in swapping and arguing back and forth about which way it should be, and these things really don't matter all that much. Instead of doing that, what you can do is you can just have Black autoformat your code. Basically, what you get out of it is a standardized format. You can just say, "Our code standard is Black," and then I would still also couple this with a linter because there are certain things that Black doesn't find. There are certain problems it still doesn't encounter. I'd still couple it with either PEP 8 (pycodestyle), but you can just have Black basically resolve the vast majority of these code formatting issues and have it autoformat your code. So you don't have todo this very manual, intensive process of fixing all these style lines, of all these code styling issues, and you don't have to get in any of these pointless debates about what the proper style is. Once again, we install Black as our standard way of using pip, and then we have a utility available to use now. Black does require Python 3 in order to do the formatting. You have to be using Python 3 to do your formatting.

You can actually format Python 2 code using Black, but you actually have to have a running Python 3 environment to use it.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pip list | grep black

black 18.9b0

From the previous example we know our code is clean but let us check if it satisfies the code standard set by black. You can see black still says it has to be reformatted.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ black --check pylama\_test.py

would reformat pylama\_test.py

All done! 💥 💔 💥

1 file would be reformatted

Now before making any reformatting, I will copy the code to another file and run the code against black.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cp pylama\_test.py black\_reformat.py

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ black black\_reformat.py

reformatted black\_reformat.py

All done! ✨ 🍰 ✨

1 file reformatted

Now I know something has been changed between my two files, so let me do a diff to check it.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ diff pylama\_test.py black\_reformat.py

12,13c12,13

< "Authorization": "Token {}".format(token)

< }

---

> "Authorization": "Token {}".format(token),

> }

Pyplus - Pytest Netmiko

mercredi, 15 mai 2019

19:58

Lets start looking at what pytest means.

With programming, having testing gives you a much higher degree of comfort that your code isworking properly. It basically corresponds to how much testing you have. As you get more andmore complicated code, you basically start to lose comfort whether any of your changes aregoing to break things.Having a robust set of testing also gives you this capability that as you learn more and morethings about the behavior of your system, you can just directly incorporate them into your test suite.

Now in Python, there are really two main libraries that are frequently used, and there's more,there's definitely a whole set of other libraries. But the two biggest ones are going to be pytest and there's also going to be the builtin unittest libraries. Python has unittest, which is built inand it also has pytest.In general, I prefer pytest a lot more than unittest. I always find using unittest [is] not as nice as using pytest.

For pytest, you need to specify a function with name as test. If there is no function is defined as test, even if the file name is test, the pytest will say no tests ran. see the following, I have the file named as test\_netmiko.py

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pip list | grep pytest

pytest 4.4.0

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ ls | grep test\*

test\_netmiko.py

I am going to run pytest on the whole directory. I'm passing the ‘s’. Now by default, pytest does something special with standard output, it actually hides standard output. In this case, I actually want normal standard output to be returned because I want things that I put in to be printed so that I can see what's going on better with my test. I also do the ‘v’, for the verbose.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ py.test -s -v .

========================== test session starts =======================================

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collecting ... Password:

collected 0 items

==================== no tests ran in 3.67 seconds =========================

Now, I going to define a function with name as test\_netmiko\_find\_devicename and also I will change the file name as netmiko\_overview.py. This means my file name do not have the word test in it but the contents of that file has a function named test in it.

The function "test\_netmiko\_find\_devicename()" is going to check if there is any keyword cisco3# in the output of the variable connect\_rtr.find\_prompt(). Here is the code

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat netmiko\_overview.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

def test\_netmiko\_find\_devicename():

connect\_rtr = ConnectHandler(\*\*device1)

assert "cisco3#" in (connect\_rtr.find\_prompt())

I run the pytest against the whole directory but you can see the pytest was not able to find that function with name "test\_netmiko\_find\_devicename" inside that file because when you run a test on a whole directory, both file name and function should have the "test" keyword present.

so now instead of running the pytest against the whole directory, I am going to run it against directly against the file and you can see the test is passed.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ py.test -s -v netmiko\_overview.py

========================== test session starts ====================================

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collecting ... Password:

collected 1 item

netmiko\_overview.py::test\_netmiko\_find\_devicename PASSED

========================== 1 passed in 8.70 seconds ============================================

Now let me modify the code and see if the test fails. I have did some modification and ran the test below and you can see it failed and the reason for failure. Basically it says the assertion error and it shows instead of "cisco1#" keyword in the find prompt, "cisco3#" was displayed.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ py.test -s -v netmiko\_overview.py

=============================== test session starts ================================

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collecting ... Password:

collected 1 item

netmiko\_overview.py::test\_netmiko\_find\_devicename FAILED

============================ FAILURES========================

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ test\_netmiko\_find\_devicename \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def test\_netmiko\_find\_devicename():

connect\_rtr = ConnectHandler(\*\*device1)

> assert "cisco1#" in (connect\_rtr.find\_prompt())

E AssertionError: assert 'cisco1#' in 'cisco3#'

E + where 'cisco3#' = <bound method BaseConnection.find\_prompt of <netmiko.cisco.cisco\_ios.CiscoIosSSH object at 0x7fc14adbd0b8>>()

E + where <bound method BaseConnection.find\_prompt of <netmiko.cisco.cisco\_ios.CiscoIosSSH object at 0x7fc14adbd0b8>> = <netmiko.cisco.cisco\_ios.CiscoIosSSH object at 0x7fc14adbd0b8>.find\_prompt

netmiko\_overview.py:12: AssertionError

========================== 1 failed in 9.56 seconds =============================================

I am going to create another test function and run the pytest specifically against that function. Here is the code.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat netmiko\_overview.py

from netmiko import ConnectHandler

from getpass import getpass

device1 = {

"host": 'cisco3.lasthop.io',

"username": 'pyclass',

"password": getpass(),

"device\_type": 'cisco\_ios',

}

def test\_netmiko\_find\_devicename():

connect\_rtr = ConnectHandler(\*\*device1)

assert "cisco3" in (connect\_rtr.find\_prompt())

def test\_netmiko\_find\_dns():

connect\_rtr = ConnectHandler(\*\*device1)

dns\_server = connect\_rtr.send\_command("sh run")

assert "name-server" in dns\_server

You can see below the pytest against the specific function has worked.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ py.test -s -v netmiko\_overview.py::test\_netmiko\_find\_dns

============================= test session starts ====================================

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collecting ... Password:

collected 1 item

netmiko\_overview.py::test\_netmiko\_find\_dns PASSED

========================== 1 passed in 9.51 seconds =====================================

Pyplus - Pytest Fixtures

mercredi, 15 mai 2019

23:20

Lets look at why we need pytest fixtures. Here is my code and when you run it using pytest, to execute single script it does the SSH connection 2 times which is not correct when we use pytest.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat netmiko\_overview.py

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

def netmiko\_conn():

connect\_rtr = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = password,

device\_type = 'cisco\_ios',

)

return connect\_rtr

def test\_netmiko\_find\_devicename():

connect\_rtr = netmiko\_conn()

assert "cisco3#" in connect\_rtr.find\_prompt()

def test\_netmiko\_find\_ntp():

connect\_rtr = netmiko\_conn()

ntp\_server = connect\_rtr.send\_command("sh run")

assert "ntp server" in ntp\_server

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pytest -s -v netmiko\_overview.py

===========================test session starts =========

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collecting ... Password:

collected 2 items

netmiko\_overview.py::test\_netmiko\_find\_devicename PASSED

netmiko\_overview.py::test\_netmiko\_find\_ntp PASSED

============================ 2 passed in 14.10 seconds =======

You can see there is 2 separate SSH connection to the same device

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ netstat -an | grep 184.10

tcp 0 0 172.30.0.144:43902 184.105.247.89:22 TIME\_WAIT

tcp 0 0 172.30.0.144:43900 184.105.247.89:22 TIME\_WAIT

Each one of these two test functions, the‘test\_find\_prompt’ and the ‘test\_send\_command’ have [each] established a separate Netmikoconnection. [If] We had 10 different functions there, we would go through this process ofestablishing Netmiko connection 10 different times. This question arises, “How do we establish a Netmiko connection only once?” “How do we reuse that connection across multiple of thesetest functions.

Pytest fixtures solves this problem . See this script, I will say ‘import pytest’. Now I've added these "@pytest.fixture". We then say we're going to define this ‘netmiko\_conn()’ as a fixture. Now the default for a fixture is, it gets reused every time. The default for how frequently it gets reused is at the function level. Let me first show you how to reuse the fixture, and then I'll show you how to change the scope of the fixture to module level.

Now what we do is, in each one of our tests where we want to use that fixture, we actually passin the fixture name. Here I pass in the fixture name. What pytest is going to do is it's going tosearch in a certain way to find this fixture name, and actually in this case, it sees we've registered it here in this module that we're using. It's then going to execute the fixture code, andit's going to assign into the variable this ‘netmiko\_conn’, whatever is returned from this fixture. Inthis case, we're no longer going to have to call this in this way, and then we're going to need tochange this name here because the Netmiko connection is now going to be known as this,whatever is returned from that fixture named ‘netmiko\_conn’ is going to be the Netmikoconnection.We go and do the same process here. We're going to use our fixture name, which is‘netmiko\_conn’, which we've registered above using this app ‘pytest.fixture’ decorator. Thenonce again, our Netmiko connection is now going to be known as whatever that fixture namewas, as that's what we're returning with that fixture. Now we'll run this. We'll see whether it stillruns. Now we're using this fixture.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat fixtures.py

import pytest

from netmiko import ConnectHandler

from getpass import getpass

**@pytest.fixture**

def netmiko\_conn():

connect\_rtr = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = getpass(),

device\_type = 'cisco\_ios',

)

return connect\_rtr

def test\_netmiko\_find\_devicename(netmiko\_conn):

assert "cisco3#" in netmiko\_conn.find\_prompt()

def test\_netmiko\_find\_ntp(netmiko\_conn):

ntp\_server = netmiko\_conn.send\_command("sh run")

assert "ntp server" in ntp\_server

The fixture, at this point, will still be establishing two separate SSH connections. First, let's verify that our code still runs, and that we are in fact establishing two SSH connections. Now the default scope of this fixture is it gets reused once per function. We call it here, and it'sgoing to set up a new SSH connection. This other function will not reuse the existing connection and it has to go and make a new one

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pytest -s -v fixtures.py

======================test session starts ======================

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collecting ... Password:

collected 2 items

fixtures.py::test\_netmiko\_find\_devicename PASSED

fixtures.py::test\_netmiko\_find\_ntp PASSED

=================== 2 passed in 14.33 seconds ====================

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ netstat -an | grep 184.10

tcp 0 0 172.30.0.144:43896 184.105.247.89:22 TIME\_WAIT

tcp 0 0 172.30.0.144:43894 184.105.247.89:22 TIME\_WAIT

If we had five other test functions, it would reuse it onceeach time, but we can change that behavior by changing the scope of the fixture. Now I'm goingto change the scope from being at the function level to say it's at the module level. Anytime, anything inside this one specific Python file uses this ‘netmiko\_conn’, it's going to reuse it.Every time it gets reused inside this file, it's going to reuse the existing connection. First time itgets called here, it will establish the connection, but the second time it gets reused here, it'sgoing to reuse this fixture because they're all shared with inside this same Python module.

py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat fixtures.py

import pytest

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

**@pytest.fixture(scope="module")**

def netmiko\_conn():

connect\_rtr = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = password,

device\_type = 'cisco\_ios',

)

return connect\_rtr

def test\_netmiko\_find\_devicename(netmiko\_conn):

assert "cisco3#" in netmiko\_conn.find\_prompt()

def test\_netmiko\_find\_ntp(netmiko\_conn):

ntp\_server = netmiko\_conn.send\_command("sh run")

assert "ntp server" in ntp\_server

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pytest -s -v fixtures.py

=============================== test session starts ===============

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collected 2 items

fixtures.py::test\_netmiko\_find\_devicename Password:

PASSED

fixtures.py::test\_netmiko\_find\_ntp PASSED

==============================2 passed in 9.82 seconds =============

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ netstat -an | grep 184.10

tcp 0 0 172.30.0.144:43882 184.105.247.89:22 TIME\_WAIT

Now when we ran it, we should see that only one SSH connection gets created for these two different tests. We can see we only had one SSH connection happen there. So we nowhave a fixture that lets us share this SSH connection across multiple tests, and you could obviously create multiple fixtures.

Pyplus - Pytest Fixtures teardown

mardi, 28 mai 2019

12:22

In our previous example, we established an SSHconnection using Netmiko. But, we never gracefully tore that SSH connection down at the end ofthe given set of tests. In our case, this was our scope module. So this was for this entire Pythonfile. Now let's look at a graceful teardown of a Netmiko SSH connection.Here we have our ‘@pytest.fixture’ decorator, where we're registering this as a fixture. Now,notice I've added an argument. This ‘request’ argument. What we're going to do here at the endis we're going to basically do this ‘addfinalizer’ [request.addfinalizer(teardown\_ssh)]. This ‘addfinalizer’ is going to get called when the given set of tests are done, and then we specify something that itneeds to call.Here we specify this function named ‘teardown\_ssh’ and inside that function, we've defined what we want to happen, and in this case, we're going to call the Netmiko connection and we're going to specifythat we need to execute disconnect on it. Basically, we've defined an action that we want to happen when our series of tests are done and in this case this is this SSH disconnect operation[def teardown\_ssh(): net\_connect.disconnect()].

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat fixtures.py

import pytest

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

@pytest.fixture(scope="module")

def netmiko\_conn(request):

connect\_rtr = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = password,

device\_type = 'cisco\_ios',

)

def teardown\_ssh():

connect\_rtr.disconnect()

request.addfinalizer(teardown\_ssh)

return connect\_rtr

def test\_netmiko\_find\_devicename(netmiko\_conn):

assert "cisco3#" in netmiko\_conn.find\_prompt()

def test\_netmiko\_find\_ntp(netmiko\_conn):

ntp\_server = netmiko\_conn.send\_command("sh run")

assert "ntp server" in ntp\_server

We'll go ahead and run this script, setting up our SSH connection, executing the ‘find\_prompt’ method. Here in a second, we're executing ‘send\_command’, then we should have gracefully disconnected.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pytest -s -v fixtures.py

======================test session starts ============================

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collecting ... Password:

collected 2 items

fixtures.py::test\_netmiko\_find\_devicename PASSED

fixtures.py::test\_netmiko\_find\_ntp PASSED

====================== 2 passed in 9.48 seconds ======================

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ netstat -an | grep 184.105

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ netstat -an | grep 184.105

Now, here we actually see that we have nothing showing up in netstat because when we gracefully disconnect, the TCP/SSH connection gracefully goes away, so it doesn't show up inthe TCP state table in netstat any longer. We're gracefully closing our SSH connection using this ‘addfinalizer’. So, think of one of them being a set up process, that's what I showed your firstwith the fixture. This is a tear down process, what happens at the end of a series of tests.

Pyplus - Pytest with conftest

mardi, 28 mai 2019

13:30

It would be nice if we had a way to centralize the definition of our fixtures, so that wecould have fixtures that were reused across multiple test modules. pytest does give us a way todo this. A way we can do this as using this ‘conftest.py’ file. What pytest does is, it's going to automatically look through ‘conftest.py’ to find any fixture definitions.

I am going to move the fixtures and import statement into a file named "conftest.py".

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat conftest.py

import pytest

from netmiko import ConnectHandler

from getpass import getpass

password = getpass()

@pytest.fixture(scope="module")

def netmiko\_conn(request):

connect\_rtr = ConnectHandler(

host = 'cisco3.lasthop.io',

username = 'pyclass',

password = password,

device\_type = 'cisco\_ios',

)

def teardown\_ssh():

connect\_rtr.disconnect()

request.addfinalizer(teardown\_ssh)

return connect\_rtr

And then have only the test function on the script "test\_netmiko.py" that we are going to execute.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ cat test\_netmiko.py

def test\_netmiko\_find\_devicename(netmiko\_conn):

assert "cisco3#" in netmiko\_conn.find\_prompt()

def test\_netmiko\_find\_ntp(netmiko\_conn):

ntp\_server = netmiko\_conn.send\_command("sh run")

assert "ntp server" in ntp\_server

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-Pytools]$ pytest -s -v test\_netmiko.py

Password:

============ test session starts =============

platform linux -- Python 3.6.8, pytest-4.4.0, py-1.8.0, pluggy-0.9.0 -- /home/sethuvignesh/VENV/py3\_venv/bin/python36

cachedir: .pytest\_cache

rootdir: /home/sethuvignesh/Viki/Class-Pytools

plugins: pylama-7.6.6

collected 2 items

test\_netmiko.py::test\_netmiko\_find\_devicename PASSED

test\_netmiko.py::test\_netmiko\_find\_ntp PASSED

==============2 passed in 5.90 seconds ==========

Pyplus - **XML - Why Care?**

mardi, 28 mai 2019

15:49

In this lesson we're going to talk about several aspects of XML. The first question we need to talk about with respect to XML is why do we care about XML? Why do we care about parsing XML? Why can't we just use JSON and totally dispense with worrying about, caring about XML?The general answer is there are several contexts where we still have XML, or XML is still goingto be used under the hood, so we might need to know about it. We might need to handle it in some special way. What are these contexts? NETCONF still fundamentally uses XML under thehood for how the data is serialized. XML is still used in NETCONF. We also have several APIs like NXAPI that are using, or have the capabilities of using, XML. Similarly, a lot of networking devices have the capability, when your SSH’ed in, to pipe commands out into XML format.You can do piped XML, pipe display XML. There are going to be variations of this. Also on platforms like Cisco, IOSXR.On IOSXR, you're going to have an XML subsystem. You can dothings like go into XML agent and send in XML commands, and get XML output back to you.You can rightfully ask, "Okay, well a lot of these things have newer versions, like NETCONF hasa RESTCONF. NXAPI has a JSON mode, a JSONRPC mode. IOS XR has gRPC. Why do wes till need to do XML? Why do we still need to care about XML?" The bottom line is we have a lot of field devices. The support for these newer things is in varying levels of how many commands they support.

Do you have these features available to you, like gRPC, RESTCONF? How good is that newerfeature on the given platform? In a lot of cases, XML is going to be the more widely available,the better solution to use to try to support your set of field devices.Now that we've come to the conclusion that we're going to need to potentially support XML, thatwe're going to need to understand how to use XML and Python, there's a question of whichlibraries in Python do we need to care about? Which libraries do we need to worry about? Ingeneral, we're going to be talking about three things in this lecture as far as Python libraries.We're going to be talking about XML to dict. How do we convert XML more like it was JSON,and convert it over to dictionary structure. We're also going to be talking about lxml, and we'regoing to be talking about the builtin XML library to a certain extent using ElementTree.

Pyplus - XML Introduction

mardi, 28 mai 2019

15:50

Let's talk about how to think about XML andhow to model it so we can understand it better, so we can understand how to deal with it betterprogrammatically.If you look at what I have here on my screen, you're going to see that I actually have this XMLoutput from a Juniper device. This is the ‘show version | display xml’ command. You can seethat I get this structure back. Now the key thing to notice is that there's going to be a hierarchyto this structure. Here we have these ‘rpcreply’ tags. Inside the ‘rpcreply’ tags, at the nextlevel, we have this ‘softwareinformation’ tag, and we also have this ‘cli’ tag. If we dig further intothat, we have this ‘hostname’, ‘productmodel’, ‘productname’, ‘jsr/’, ‘packageinformation’ tagsinside ‘softwareinformation’.

We have this structure to XML, and this structure could be modeled as a tree. In general, whenwe're going to use XML we're going to want to think about it as a tree. Now, you might heararguments, if you read about XML, that it's treelike. There might be certain reasons why it's nottechnically a tree, but for our purposes I think we can generally regard it and treat it as a tree.

What do I mean by a tree? What I mean by a tree is we have a set of nodes. There's going to besome root node, which is going to be the top of the tree. Underneath the root node, there'sgoing to be child nodes, and these child nodes belong to the root node.

The characteristic of the child node is they're only going to have one parent. If we go back andlook at our structure here, we can see our ‘rpcreply’ is our root node and we can see that it haschildren of ‘softwareinformation’ and ‘cli’. If you asked a question of, what's the parent of‘softwareinformation’? There is only one answer to it, it's ‘rpcreply’ node. Similarly, if you askeda question of what's the parent of ‘hostname’, there's only going to be only answer to it. It'sgoing to be ‘softwareinformation’, and ‘productmodel’ has the same parent.

We have this tree structure where we have a root node and we have child nodes and if we digdown into it more there's going to be children of ‘softwareinformation’ ‘hostname’ is a child of‘softwareinformation’. The ‘productmodel’ is a child of ‘softwareinformation’. The productname’ is a child of ‘softwareinformation’. Now, when we think of this, these nodes aregoing to be known as Element Nodes. If we go back and look at our structure, you can see herethat we have a tag called ‘rpcreply’. This will make an Element Node. We have a tag of‘softwareinformation’. This will make an Element Node. We have a tag of ‘hostname’. This willmake an Element Node.

We're going to have these nodes, they're going to form the tree. We're going to form a treeusing the tags. Now, when we look at this some more, we see not only do we have tags in the XML, but we have text fields. You can see here that when we dig down into this for the hostname, we have this text field of ‘pynetjnprsrx1’, i.e. this is the ‘hostname’ on this Juniperdevice and we have a ‘productmodel’ of ‘srx100h2’.These text fields are not tags, they're actually the data, in this case, between the tags. The datahere in between this ‘productmodel’ tags. You might frequently see this referred to as being atext node and I put it in quotes and I'll talk about this distinction here in a minute of why I put it inquotes.

In Python, we're fundamentally, in most cases, at least most cases that I'm going to talk about inthis lesson, we're going to use lxml or the builtin Python library to be processing this XML.When we use this library, we're only going to treat Elements as nodes.We're only going to treat our tags as nodes. We are not going to treat the text as being nodes inthe tree. I'll talk about the implications of this in a little bit, but when we're dealing this in lxmllibrary or in the builtin XML library, we're going to want to think of this text as being an attribute,as being associated with this Element Node. Here I have my ‘hostname’ Element Node and ithas some text associated with it. Similarly, I have my ‘productmodel’ Element Node, and it hassome text associated with it. The text here is not going to form a separate node in the tree. Theonly nodes in the tree that we're going to have from an Element tree in lxml perspective is goingto be the tags these Element Nodes. Similar to this, we can also have attributes in our XMLand here I just made one up. I made this ‘fake\_attribute=’whatever’’ and notice that this is insideour tag. Here we have our ‘productmodel’ tag, then I add an attribute to it, and I gave it a valueof ‘whatever’.

When I go and look at that, the way to consider this once again is to make these attribute, be anattribute of the Element Node. So think of it as being associated with the Element Node. HereI'm using attribute two ways. It's an attribute from the XML perspective and it's also beingconsidered an attribute in the Python perspective. We're not going to consider these attributes as being separate nodes in the tree. I've been making this distinction of nodes of how they'reviewed in lxml and ElementTree because this perspective on the XML is different than theperspective that is frequently associated within the browser and the DOM. In the browser wehave this thing called the DOM, the domain object model, and it has a way of treating XML andin it’s way of viewing the tree, of this XML tree, it does consider Elements as nodes and text asnodes and attribute as nodes.

They are their own object in the tree, but we're not going to be doing this in Python. The nodesthat we're going to have in Python are these Element Nodes. Then you can rightfully ask, "Okay,well, why the heck do I care about all these distinctions? Why am I going to are about this?"Well, when you do something in Python and you have this object this Element Node, like ‘hostname’, and you answer a question like, "Get me the children. Get me the nodes that belong tothis hostname," you're going to have different answers depending on how you model this tree. We're modeling this tree only as Element Nodes. So when we say, "Get the children of ‘hostname’," it's only going to get the Element Node, children of this. When I say, "Get me thechildren of ‘softwareinformation’," it's going to get me the ‘hostname’, the ‘productmodel’, the‘productname’, potentially other things.

There are implications in what this means to your programming, and if we step back at a higherlevel for a second, you can clearly see we have something that's different here than just JSON.With JSON, we just have lists and dictionaries. Here we have something more than lists anddictionaries. We have a hierarchical tag structure, we potentially have text fields inside thishierarchical tag structure and we also potentially have these attributes associated with a tag.This is not going to directly map to Python lists and dictionaries as easily as it maps in JSON.

Pyplus - xmltodict library

lundi, 3 juin 2019

18:12

We're going to start talking about XML to dictionary, which is a Python library thatcan be useful in trying to handle XML. XML to dictionary library basically is just trying to convertXML straight into a Python dictionary.

First thing is to pip install the library xmltodict.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Viki]$ pip install xmltodict

Requirement already satisfied: xmltodict in /home/sethuvignesh/VENV/py3\_venv/lib/python3.6/site-packages (0.11.0)

I have a test file named "junos\_shver.xml"

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-XML]$ ipython

Python 3.6.8 (default, May 24 2019, 18:27:52)

Type 'copyright', 'credits' or 'license' for more information

IPython 7.2.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: xml\_file = open("junos\_shver.xml")

First I will read the file and then stripout any leading trailing whitespace

In [2]: xml\_data = xml\_file.read().strip()

In [3]: print(xml\_data)

<software-information>

<host-name>pynet-jnpr-srx1</host-name>

<product-model>srx100h2</product-model>

<product-name>srx100h2</product-name>

<jsr/>

<package-information>

<name>junos</name>

<comment>JUNOS Software Release [12.1X44-D35.5]</comment>

</package-information>

</software-information>

In [4]: import xmltodict

In [5]: xml\_dict1 = xmltodict.parse(xml\_data)

In [6]: xml\_dict1

Out[6]:

OrderedDict([('software-information',

OrderedDict([('host-name', 'pynet-jnpr-srx1'),

('product-model', 'srx100h2'),

('product-name', 'srx100h2'),

('jsr', None),

('package-information',

OrderedDict([('name', 'junos'),

('comment',

'JUNOS Software Release [12.1X44-D35.5]')]))]))])

In [7]: xml\_dict1.keys()

Out[7]: odict\_keys(['software-information'])

In [9]: xml\_dict1["software-information"]

Out[9]:

OrderedDict([('host-name', 'pynet-jnpr-srx1'),

('product-model', 'srx100h2'),

('product-name', 'srx100h2'),

('jsr', None),

('package-information',

OrderedDict([('name', 'junos'),

('comment',

'JUNOS Software Release [12.1X44-D35.5]')]))])

In [10]: xml\_dict1["software-information"].keys()

Out[10]: odict\_keys(['host-name', 'product-model', 'product-name', 'jsr', 'package-information'])

In [11]: xml\_dict1["software-information"]["product-name"]

Out[11]: 'srx100h2'

In [12]: xml\_dict1["software-information"]["package-information"]

Out[12]:

OrderedDict([('name', 'junos'),

('comment', 'JUNOS Software Release [12.1X44-D35.5]')])

In [13]: xml\_dict1["software-information"]["package-information"].keys()

Out[13]: odict\_keys(['name', 'comment'])

Pyplus - xmltodict and list problem

lundi, 3 juin 2019

19:10

When there is only one tag, the element returned is a string but when there is two tags the element returned is list. This becomes a problem in our parsing code. To avoid this we will force to get the output as list irrespective of how many tags we have.

In [1]: import xmltodict

In [2]: f = open("junos\_shver.xml")

In [3]: xml\_data = f.read().strip()

In [4]: print(xml\_data)

<software-information>

<host-name>pynet-jnpr-srx1</host-name>

<product-model>srx100h2</product-model>

<product-name>srx100h2</product-name>

<product-name>srx100h3</product-name>

<jsr/>

<package-information>

<name>junos</name>

<comment>JUNOS Software Release [12.1X44-D35.5]</comment>

</package-information>

</software-information>

In [5]: my\_xml = xmltodict.parse(xml\_data)

In [6]: my\_xml

Out[6]:

OrderedDict([('software-information',

OrderedDict([('host-name', 'pynet-jnpr-srx1'),

('product-model', 'srx100h2'),

('product-name', ['srx100h2', 'srx100h3']),

('jsr', None),

('package-information',

OrderedDict([('name', 'junos'),

('comment',

'JUNOS Software Release [12.1X44-D35.5]')]))]))])

In [7]: my\_xml.keys()

Out[7]: odict\_keys(['software-information'])

In [9]: my\_xml['software-information']

Out[9]:

OrderedDict([('host-name', 'pynet-jnpr-srx1'),

('product-model', 'srx100h2'),

('product-name', ['srx100h2', 'srx100h3']),

('jsr', None),

('package-information',

OrderedDict([('name', 'junos'),

('comment',

'JUNOS Software Release [12.1X44-D35.5]')]))])

In [10]: my\_xml['software-information']['product-name']

Out[10]: ['srx100h2', 'srx100h3']

In [11]: my\_xml['software-information']['product-model']

Out[11]: 'srx100h2'

Here when you look at the product name and product model, product-name is list of strings but product-model is just string. This becomes an issue when we parse.

so we have to force that even if its one tag or more than one tag, we force it to parse and present as list of strings.

In [12]: my\_xml = xmltodict.parse(xml\_data, force\_list={'product-model': True})

In [13]: my\_xml['software-information']['product-model']

Out[13]: ['srx100h2']

In [14]: my\_xml['software-information']['product-name']

Out[14]: ['srx100h2', 'srx100h3']#

You can see if it is one element or multiple element, it is represented as list.

Pyplus - NX-API Overview

lundi, 10 juin 2019

18:46

NXAPI is an API available on the CiscoNexus platform. It's going to use HTTP/HTTPS for transport. Embedded inside this HTTPSsession, we are going to be passing XML or JSONRPC data. Right now the main Pythonlibraries that I'm using to interface to NXAPI are either pynxos or nxapiplumbing. You could also use the NAPALM library. TheNAPALM library actually uses the nxapiplumbing library under the hood, but it also providesyou an interface that abstracts away these lower level mechanics.

We will do ssh and We can do this ‘show run | inc nxapi’, and we can see We have‘feature nxapi’ configured, we've actually enabled this NXAPI Sandbox. ENow if we look at this NXAPI Sandbox, you can see here that there are a few options, and this is really only fortesting and experimentation purposes. You should have NX-API sandbox disabled on live devices as far ast he Sandbox. But you can see up here would that we have three different message formats. We can either use JSONRPC, XML, or JSON. In general, we should either be using JSONRPC or XML.

Now XML is the first format that Cisco published on the NXAPI, that’s the first message formatthey supported with NXAPI. I've also heard from an individual that works a lot with variousNexus platforms that the XML support has the longest history, and also is the least prone toproblems. When comparing the JSONRPC versus XML, he basically convinced me that wewere better off to use XML to interface to the Cisco NXOS platforms, at least as of today.

Let's just look at the JSONRPC format first. If we put in acommand, like ‘show version’, here and we do an ‘HTTP POST’, we can see that here we'reconstructing our JSONRPC request and we get JSON-RPC response. JSON-RPC response we get back is structured data.

The difference between cli and cli\_ascii command type is cli shows everything as structred but cli\_ascii shows the body as one big string.

Pyplus - NX-API and Json RPC

lundi, 17 juin 2019

17:46

Let's look at some code that interfaces to NXAPI using JSONRPC. In the following code, we run the command to get the clock of the nxos device.

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-NXAPI]$ cat nxapi-json.py

import requests

from pprint import pprint

from getpass import getpass

from urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "jsonrpc",

host = "nxos1.lasthop.io",

username = "pyclass",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

output = device.show("show clock")

pprint(output)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-NXAPI]$ python nxapi-json.py

Password:

{'simple\_time': '17:43:12.405 UTC Mon Jun 10 2019\n', 'time\_source': 'NTP'}

In this following code, we run set of show commands to get the output

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-NXAPI]$ cat nxapi-json.py

import requests

from pprint import pprint

from getpass import getpass

from urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "jsonrpc",

host = "nxos1.lasthop.io",

username = "pyclass",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cmds = ["show hostname","show ntp peers","show clock"]

output = device.show\_list(cmds)

pprint(output)

(py3\_venv) [sethuvignesh@ip-172-30-0-144 Class-NXAPI]$ python nxapi-json.py

Password:

[{'command': 'show hostname', 'result': {'hostname': 'nxos1.lasthop.io'}},

{'command': 'show ntp peers',

'result': {'TABLE\_peers': {'ROW\_peers': [{'PeerIPAddress': '152.2.21.1',

'conf\_flag': '(configured)',

'serv\_peer': 'Server'},

{'PeerIPAddress': '130.126.24.24',

'conf\_flag': '(configured)',

'serv\_peer': 'Server'}]}}},

{'command': 'show clock',

'result': {'simple\_time': '18:01:45.411 UTC Mon Jun 10 2019\n',

'time\_source': 'NTP'}}]

Now its time to configure something on the Nexus using NX-API. You need the following configuration on the Nexus for the nxapi to work.

*feature nxapi*

*nxapi https port 8443*

Let us try to configure something, for example change the "logging history size".

root@PyNetServer:~/PyPlus# cat nxapi-json.py

import requests

from pprint import pprint

from getpass import getpass

from urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "jsonrpc",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cfg\_cmd = [

"logging history size 300"

]

output = device.config\_list(cfg\_cmd)

print(output)

root@PyNetServer:~/PyPlus# python nxapi-json.py

Password:

[{u'command': 'logging history size 300', u'result': None}]

DCSW1# sh run | inc logging

logging history size 300

You can do a list of configuration commands as well

root@PyNetServer:~/PyPlus# cat nxapi-json.py

import requests

from pprint import pprint

from getpass import getpass

from urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "jsonrpc",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cfg\_cmd = [

"logging history size 200",

"logging console 5"

]

output = device.config\_list(cfg\_cmd)

print(output)

root@PyNetServer:~/PyPlus# python nxapi-json.py

Password:

[{u'command': 'logging history size 200', u'result': None}, {u'command': 'logging console 5', u'result': None}]

We could also do is there is a save method for saving the running config into the startup config

root@PyNetServer:~/PyPlus# cat nxapi-json.py

import requests

from pprint import pprint

from getpass import getpass

from urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "jsonrpc",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cfg\_cmd = [

"logging history size 300"

]

**output = device.save()**

print(output)

root@PyNetServer:~/PyPlus# python nxapi-json.py

Password:

True

You can see the device configuration has been saved.

*DCSW1# sh run | inc logging*

*logging history size 200*

*DCSW1# show startup-config | inc logging*

*logging history size 200*

There is also a way to do checkpoints and to dorollbacks. Here is the call that we could do to create a checkpoint. So we call ‘device.checkpoint’and we specify the file name for the checkpoint file we want to create, and we could create acheckpoint on the system [output = device.checkpoint(filename=”my\_checkpoint”)].

root@PyNetServer:~/PyPlus# cat nxapi-json.py

import requests

from pprint import pprint

from getpass import getpass

from urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "jsonrpc",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cfg\_cmd = [

"logging history size 300"

]

output = device.save()

print(output)

**checkpoint = device.checkpoint(filename="DCSW1-Jun17")**

*DCSW1# dir bootflash: | inc Jun*

*27948 Jun 17 16:17:31 2019 DCSW1-Jun17*

There is a method for also doing a rollback using this NXAPI plumbing library.

Pyplus - NX-API and XML

samedi, 20 juillet 2019

10:01

Now let's look into Cisco NXAPI using XML and the NXAPI plumbing libraries. Let'slook at this simple example that I have here. Basically importing the ‘request’ library just tohandle this selfsigned certificate. Here, I'm importing lxml, so I can print out XML that I getback, and ‘nxapi\_plumbing’. You notice this ‘device’ data structure is almost identical to what wehad before, except we have toggled this ‘api\_format=”xml”’. Then we just go ahead and executethis ‘device.show’. Here's our ‘device’ object. We execute the ‘show’ method on it, we executeour ‘show hostname’ command, and now we're getting XML back. We need some way todisplay the XML because we have this XML object that we get back. Then I use this‘etree.tostring’. I feed in the output. The decode I need to do because it comes back as a bytesobjects. I need to convert it over to a string and then I just print it out[print(etree.tostring(output).decode())]. Let's go ahead and execute that. Just look at what weget back.You can see there that I get this XML structure back, and then we can ultimately use XMLparsing, like ‘find’ or ‘findall’, to extract this string from the XML output.

root@PyNetServer:~/PyPlus/Class-NXAPI# cat nxapi-xml.py

import requests

from pprint import pprint

from lxml import etree

from getpass import getpass

from urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "xml",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

output = device.show("show hostname")

print(etree.tostring(output).decode())

root@PyNetServer:~/PyPlus/Class-NXAPI# python nxapi-xml.py

Password:

<output>

<body>

<hostname>DCSW1</hostname>

</body>

<input>show hostname</input>

<msg>Success</msg>

<code>200</code>

</output>

Now if we look at some of the other XML methods, so if you go to the top here, here we're doing our same imports, or similar imports, same device object. Here we're going to execute a series of commands

root@PyNetServer:~/PyPlus/Class-NXAPI# cat nxapi-xml.py

import requests

from lxml import etree

from getpass import getpass

from requests.packages.urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "xml",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cmds = ["show hostname","show users","show clock"]

output = device.show\_list(cmds)

for entry in output:

print(etree.tostring(entry).decode())

root@PyNetServer:~/PyPlus/Class-NXAPI# python nxapi-xml.py

Password:

<output>

<body>

<hostname>DCSW1</hostname>

</body>

<input>show hostname</input>

<msg>Success</msg>

<code>200</code>

</output>

<output>

<body>

<TABLE\_vrf>

<ROW\_vrf>

<vrf-name-out>management</vrf-name-out>

</ROW\_vrf>

</TABLE\_vrf>

<TABLE\_intf>

<ROW\_intf>

<intf-name>mgmt0</intf-name>

<prefix>192.168.122.246</prefix>

<ip-disabled>FALSE</ip-disabled>

<iod>2</iod>

<proto-state>up</proto-state>

<link-state>up</link-state>

<admin-state>up</admin-state>

</ROW\_intf>

</TABLE\_intf>

</body>

<input>show ip int brief vrf all</input>

<msg>Success</msg>

<code>200</code>

</output>

<output>

<body>

<simple\_time>05:51:13.</simple\_time>

</body>

<input>show clock</input>

<msg>Success</msg>

<code>200</code>

</output>

If we wanted to do just a single command using ‘raw\_text’, we could.Here's a single command, ‘show version’. We use the ‘show\_list’ command, pass in ourcommands, we say ‘raw text = true’ [output = device.show\_list(cmds, raw\_text=True)], and thatwould just basically give us XML output, but where we have the entire ‘show version’ output in one XML Element

root@PyNetServer:~/PyPlus/Class-NXAPI# cat nxapi-xml.py

import requests

from lxml import etree

from getpass import getpass

from requests.packages.urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "xml",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cmds = ["show version"]

output = device.show\_list(cmds, raw\_text=True)

print(etree.tostring(output[0]).decode())

*root@PyNetServer:~/PyPlus/Class-NXAPI# python nxapi-xml.py*

*Password:*

*<output>*

*<body>Cisco Nexus Operating System (NX-OS) Software*

*TAC support: <http://www.cisco.com/tac>*

*Documents: <http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home.html>*

*Copyright (c) 2002-2016, Cisco Systems, Inc. All rights reserved.*

*The copyrights to certain works contained herein are owned by*

*other third parties and are used and distributed under license.*

*Some parts of this software are covered under the GNU Public*

*License. A copy of the license is available at*

*<http://www.gnu.org/licenses/gpl.html>.*

*NX-OSv is a demo version of the Nexus Operating System*

*Software*

*loader: version N/A*

*kickstart: version 7.3(0)D1(1)*

*system: version 7.3(0)D1(1)*

*kickstart image file is: bootflash:///titanium-d1-kickstart.7.3.0.D1.1.bin*

*kickstart compile time: 1/11/2016 16:00:00 [02/11/2016 10:30:12]*

*system image file is: bootflash:///titanium-d1.7.3.0.D1.1.bin*

*system compile time: 1/11/2016 16:00:00 [02/11/2016 13:08:11]*

*Hardware*

*cisco NX-OSv Chassis ( NX-OSv Supervisor Module )*

*QEMU Virtual CPU version 2.5 with 3064740 kB of memory.*

*Processor Board ID TM6FDF0000B*

*Device name: DCSW1*

*bootflash: 3184776 kB*

*Kernel uptime is 0 day(s), 1 hour(s), 49 minute(s), 6 second(s)*

*plugin*

*Core Plugin, Ethernet Plugin*

Active Package(s)

</body>

<input>show version</input>

<msg>Success</msg>

<code>200</code>

</output>

We could also do config commands. Here you see a configuration command[cfg\_cmd]. We execute that using the config list method, we pass in our config command [output= device.config\_list(cfg\_cmd)]. Once again, we print out our XML object as a string, using this‘etree.tostring

root@PyNetServer:~/PyPlus/Class-NXAPI# cat nxapi-xml.py

import requests

from lxml import etree

from getpass import getpass

from requests.packages.urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "xml",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cmds = ["logging history size 200"]

output = device.config\_list(cmds)

print(etree.tostring(output[0]).decode())

root@PyNetServer:~/PyPlus/Class-NXAPI# python nxapi-xml.py

Password:

<output>

<body/>

<code>200</code>

<msg>Success</msg>

</output>

*DCSW1# sh run | inc history*

*logging history size 200*

Finally, we still have things like ‘device.save’ to save the configuration. We have a way to create a checkpoint [output = device.checkpoint(filename=”my\_checkpoint2”)] and we have away to do a rollback. So very similar to what we had seen with the JSONRPC, but now we're doing this using XML. Let's go ahead and execute that because that will have the configuration command and the‘save’ command executed as part of this program. Here's actually are ‘save’ command, and it just comes back as this simple XML that says it was a ‘Success’. Then here was our checkpoint creation, Here you can see what it looks like if we're using XML and this NX-API plumbing library.

root@PyNetServer:~/PyPlus/Class-NXAPI# cat nxapi-xml.py

import requests

from lxml import etree

from getpass import getpass

from requests.packages.urllib3.exceptions import InsecureRequestWarning

from nxapi\_plumbing import Device

requests.packages.urllib3.disable\_warnings(category=InsecureRequestWarning)

device = Device (

api\_format = "xml",

host = "DCSW1",

username = "cisco",

password = getpass(),

transport = "https",

port = 8443,

verify = False

)

cmds = ["logging history size 300"]

output = device.config\_list(cmds)

print(etree.tostring(output[0]).decode())

output = device.save()

print(output)

output = device.checkpoint(filename="cfg\_save\_Jul21")

print(output)

root@PyNetServer:~/PyPlus/Class-NXAPI# python nxapi-xml.py

Password:

<output>

<body/>

<code>200</code>

<msg>Success</msg>

</output>

True

None

*DCSW1# dir bootflash: | inc Jul21*

*27948 Jul 21 07:37:41 2019 cfg\_save\_Jul21*