
Practical Network Programmability
KOVARUS
Integrated IT Expertise


Agenda
• Set the stage
• Lets learn a little python
• Lets learn some Nexus
• Lets build something useful
KOVARUS
Integrated IT Expertise

Workshop Approach

- Going to start from zero with Python
- Will introduce python language concepts as we go
- If you get lost somewhere ask a question
- Feel free to follow along
- All of the code will be available on github here: https://github.com/kovarus/practical-network-programmability

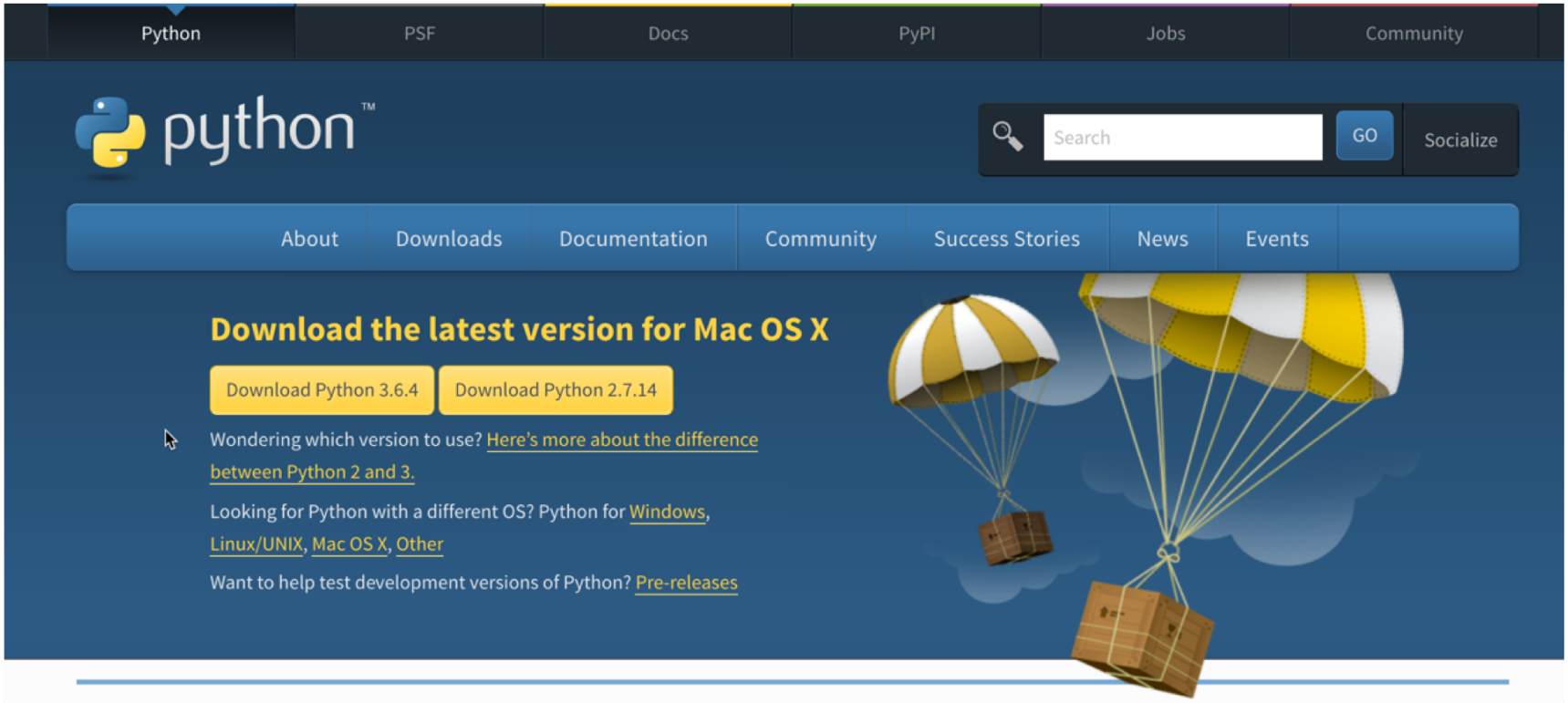
Network Programmability

- Lots of options
- NXAPI
- NETCONF/YANG
- Etc, etc. etc
- This focuses on NX-API because its easy for experienced network engineers to pick up

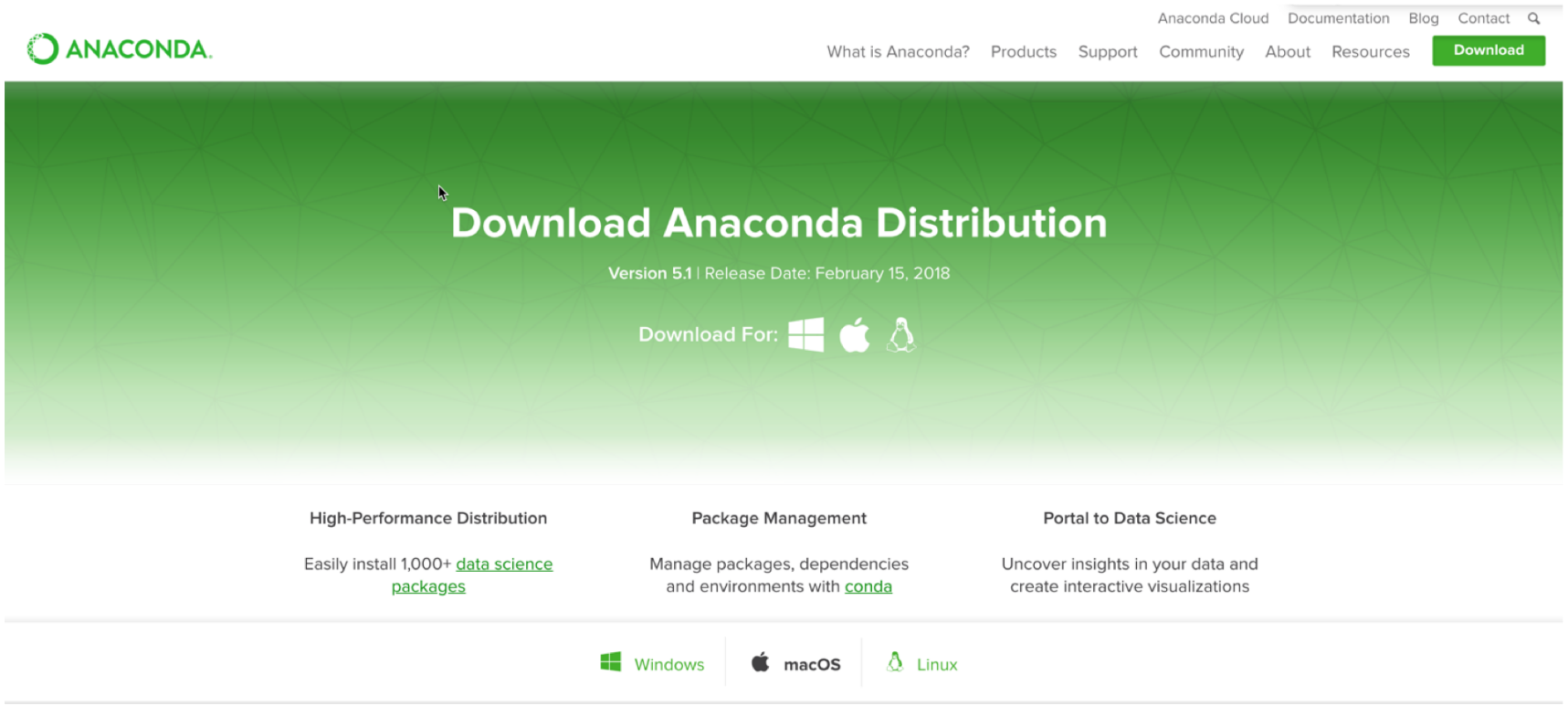
Building a Sandbox

- Going to use the Nexus 9000v for most of the tests in here
- I'm using Python3 but everything should work fine in Python 2
  - If you're on windows you can download Python from python.org or use Anaconda
  - If you're on Linux it's probably already there but you can use yum or apt-get to make sure
  - If you're on a Mac it's also probably already there but I like to use 'brew' to keep it up to date

Get Python at www.python.org



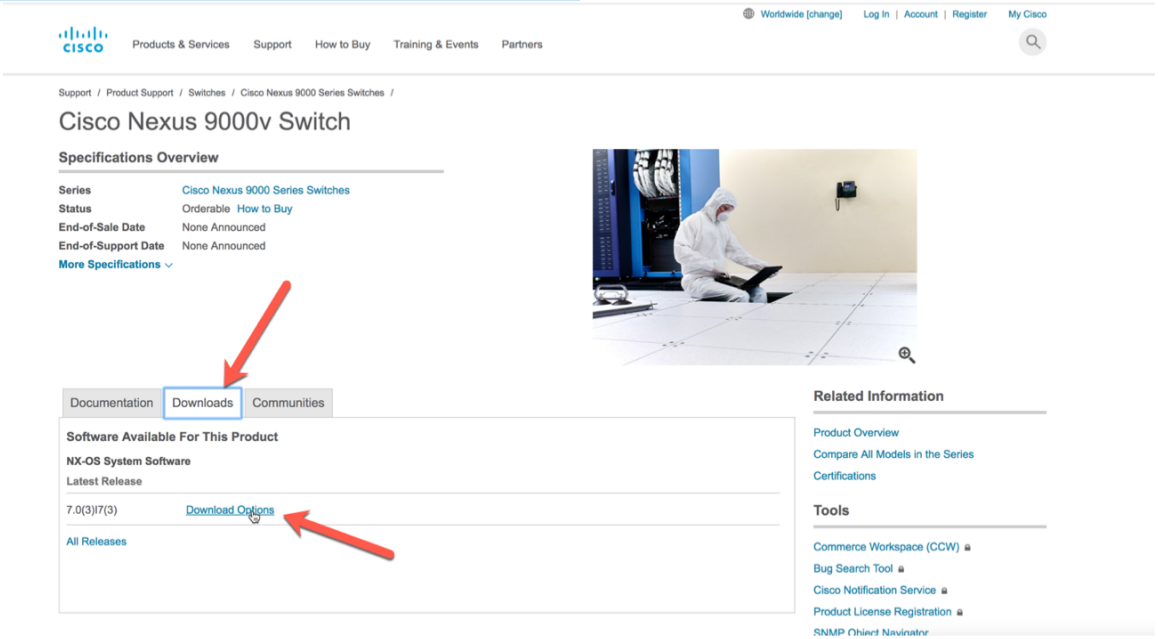
Or use the Anacoda distribution



KOVARUS
Integrated IT Expertise

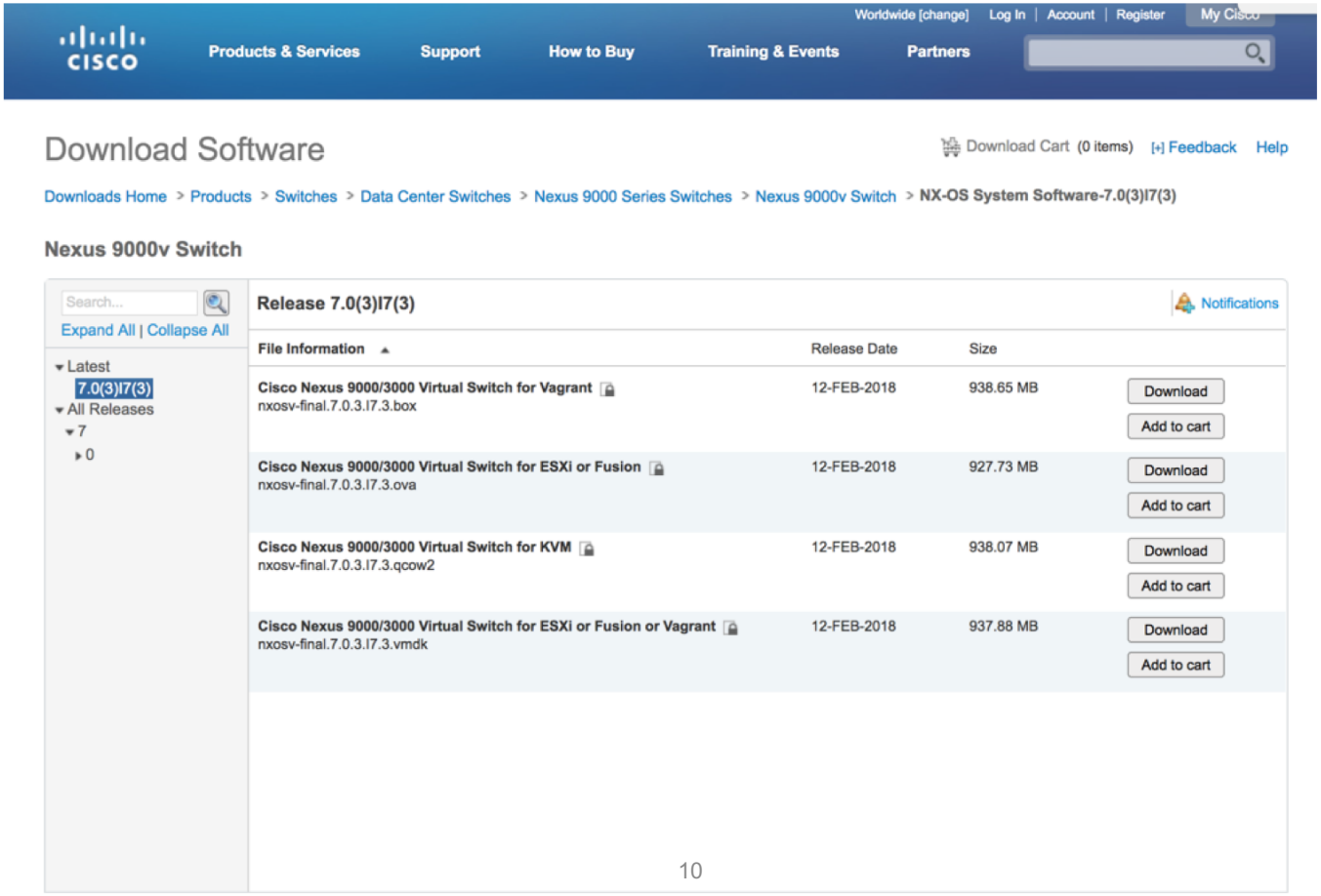
Getting a switch to test with

- https://www.cisco.com/c/en/us/support/switches/nexus-9000v-switch/model.html



KOVARUS Integrated IT Expertise

Which one?



KOVARUS
Integrated IT Expertise

Turn on the NX-API

- https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus9000/sw/6-x/programmability/guide/b_Cisco_Nexus_9000_Series_NX-OS_Programmability_Guide/b_Cisco_Nexus_9000_Series_NX-OS_Programmability_Configuration_Guide_chapter_0101.html
- (or just type ‘feature nxapi’)

A Good Text Editor is a Must

- SublimeText
- Atom
- Notepad++
- Pycharm
- Vim
- EMACS
- Doesn’t really matter as long as you’re comfortable. It should have intellisense, syntax highlighting and be responsive

Virtualenv. An optional way to make life easier

- A handy tool to let us have separate environments isolated from one another without needing a VM or container
- % virtualenv nxapiworkshop –p /usr/bin/python3
- % source nxapiworkshop/bin/activate
- We can install packages without affecting the system (pip install)
- Helps us isolate our libraries for when we share code with others! (pip freeze)


KOVARUS
Integrated IT Expertise
Some Python Basics

Why Python?

- It’s easy to learn for a lot of people
- It’s mostly human readable
- Uses dynamic types
  - To create a string you just declare it with string content!
    - my_variable = “Some string here!”
- Batteries Included! Lots of libraries out there to do a lot of the hard stuff

Some General Guidelines

- Be consistent when you code!
- If you create your strings with double quotes then always do that!
- Indentation matters
  - You must be consistent


TABS OR SPACES?
With Python we're going to use spaces as that's the community accepted style

Invoking Python

```
▶ python
Python 3.6.1 (default, Apr  4 2017, 09:40:51)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = '1'
>>> y = '1'
>>> z = x+y
>>> print(z)
11
>>> x = 1
>>> y = 1
>>> z = x+y
>>> print(z)
2
>>>
```

What's a string?

- A string is a collection of characters typically enclosed in "" or ''
- Anything can be a string even numbers as long as they're in quotes
  - some_variable = "10" <---- This is a string
  - other_variable = 10 <--- This is not
- Adding 2 strings will concatenate them together
- IP Addresses you'll typically want to be strings

What's an Int?

- An int is an integer
- If you add them together it will do the math and give the result
  - x = 1
  - y = 1
  - z = x+y
  - z will store 2

Lets talk Lists

- Sometimes you have more than 1 thing you want to store
- Lists can be created by comma separating your list items inside of []'s
  - my_routers = ["10.1.32.1", "192.168.44.254", "r3", "r4]
  - Python can count everything in a list with len(my_routers)
  - We can display everything in it with print(my_routers)

A Segue into Loops

- Each item/element in a list can be accessed by looping through the list
  - for router in my_routers:
  - print(router)
  - Prints out all of our routers
- We might use this to feed a function that does something useful:
  - for router in my_routers:
  - backup_config_to_ftp(router)

Putting the pieces together: Functions

Why Functions?

- My goal when writing code is to have as little of it as possible
- Functions allow for re-use and make code easier to follow
- DO A SMALLER EXAMPLE HERE THAT'S MUCH SIMPLER
  - USE RESULT FOR SOMETHING ELSE

Anatomy of a Function

```
1   import requests
2
3   def get_routes(switchuser, switchpassword, url):
4       myheaders = {'content-type': 'application/json'}
5       payload = {
6         "ins_api": {
7           "version": "1.0",
8           "type": "cli_show",
9           "chunk": "0",
10          "sid": "1",
11          "input": "show ip route",
12          "output_format": "json"
13        }
14      }
15      response = requests.post(url,
16                               data=json.dumps(payload),
17                               headers=myheaders,
18                               auth=(switchuser, switchpassword)).json()
19      return response
20
21  print(get_routes(switchuser='admin', switchpassword='cisco123!', url='http://192.168.49.10/ins'))
22
```

Define the function with it's input parameters

Anatomy of a Function

```
1   import requests
2
3   def get_routes(switchuser, switchpassword, url):
4       myheaders = {'content-type': 'application/json'}
5       payload = {
6         "ins_api": {
7           "version": "1.0",
8           "type": "cli_show",
9           "chunk": "0",
10          "sid": "1",
11          "input": "show ip route",
12          "output_format": "json"
13        }
14      }
15      response = requests.post(url,
16                               data=json.dumps(payload),
17                               headers=myheaders,
18                               auth=(switchuser, switchpassword)).json()
19      return response
20
21  print(get_routes(switchuser='admin', switchpassword='cisco123!', url='http://192.168.49.10/ins'))
22
```

Create our function content

Anatomy of a Function

```
1   import requests
2
3   def get_routes(switchuser, switchpassword, url):
4       myheaders = {'content-type': 'application/json'}
5       payload = {
6         "ins_api": {
7           "version": "1.0",
8           "type": "cli_show",
9           "chunk": "0",
10          "sid": "1",
11          "input": "show ip route",
12          "output_format": "json"
13        }
14      }
15      response = requests.post(url,
16                               data=json.dumps(payload),
17                               headers=myheaders,
18                               auth=(switchuser, switchpassw
19      return response
20
21  print(get_routes(switchuser='admin', switchpassword='cisco
22
```

If we want to access the results from running the function we need to return them

Anatomy of a Function

```
1   import requests
2
3   def get_routes(switchuser, switchpassword, url):
4       myheaders = {'content-type': 'application/json'}
5       payload = {
6         "ins_api": {
7           "version": "1.0",
8           "type": "cli_show",
9           "chunk": "0",
10          "sid": "1",
11          "input": "show ip route",
12          "output_format": "json"
13        }
14      }
15      response = requests.post(url,
16                               data=json.dumps(payload),
17                               headers=myheaders,
18                               auth=(switchuser, switchpassw
19      return response
20
21  print(get_routes(switchuser='admin', switchpassword='cisco123!', url='http://192.168.49.10/ins'))
22
```

We can invoke the function with our arguments

```python
import requests

def get_routes(switchuser, switchpassword, url):
    myheaders = {'content-type': 'application/json'}
    payload = {
      "ins_api": {
        "version": "1.0",
        "type": "cli_show",
        "chunk": "0",
        "sid": "1",
        "input": "show ip route",
        "output_format": "json"
      }
    }
    response = requests.post(url,
                             data=json.dumps(payload),
                             headers=myheaders,
                             auth=(switchuser, switchpassword)).json()
    return response

switchuser = "admin"
switchpassword = "!Passw0rd"

switches = ["http://192.168.49.10/ins", "http://192.168.49.11/ins"]

for switch in switches:
    print(get_routes(switchuser=switchuser, switchpassword=switchpassword, url=switch))
```

Here we create a list of switches and store our credentials in a variable. We then loop over the items in our list to execute get_routes() on them

Lets talk about scope!

- Anything you define in a function is local to the function
- That's why we have to return things out of a function
- Also true of loops and conditionals


KOVARUS
Integrated IT Expertise
Lets Talk About REST

What's an API?

- Lets cover what an API actually is here
- ABSTRACTIONS, etc.

What is REST?

- Representational State Transfer!
- What does that even mean?
  - There's no state between the client (the requesting guy) and the server (usually your device with the answers)
  - No dependencies for subsequent requests on previous requests
  - Usually done over HTTP with POSTs, PUTs, GETs, PATCHes, and DELETEs
  - Typically easy-ish to use compared to SOAP or RPC

Accessing Resources

- Every REST API is different
- Some have URLs to different functions like https://mydevice.local/api/v2/interfaces
- Others (like NXAPI) use a single path and the data we send determines functionality like https://mynexus.local/ins

Access a REST API from Python

- We'll need a python module to access the API
- A module is python that's already been written for you to make life easier
- So lets install the requests library!

We need another module to read the responses

- Python has a module called ‘json’ which helps us convert JSON into python dictionaries

More on the Nexus

- Nexus provides us an API sandbox to make life easy

Nexus API Sandbox

- First things first lets turn it on!

```
version 7.0(3)I6(1)
switchname sw1
vdc sw1 id 1
  limit-resource vlan minimum 16 maximum 4094
  limit-resource vrf minimum 2 maximum 4096
  limit-resource port-channel minimum 0 maximum 511
  limit-resource u4route-mem minimum 128 maximum 128
  limit-resource u6route-mem minimum 96 maximum 96
  limit-resource m4route-mem minimum 58 maximum 58
  limit-resource m6route-mem minimum 8 maximum 8

cfs eth distribute
feature ospf
feature interface-vlan
feature hsrp
feature lacp
feature vpc

no password strength-check
username admin password 5 $5$vmS17wzi$aQzybNo9c23BD0uB2ePyw0isGMPw5yBPsiTIG07C9i8  role network-admin
username ansible password 5 $5$NVw/MDG4$JjCKa0n.tBUd7.aDg6fDX192gumZQvFwchn8vaboWAA  role network-admin
ssh login-attempts 10

ssh key rsa 2048
ip domain-lookup
snmp-server user admin network-admin auth md5 0x420d4172dbc5afa934e5adbfd24822b6 priv 0x420d4172dbc5afa934e5adbfd24822b6 localizedkey
snmp-server user ansible network-admin auth md5 0x7c96dc428db3aa9c9f3620e270a784c1 priv 0x7c96dc428db3aa9c9f3620e270a784c1 localizedkey
rmon event 1 description FATAL(1) owner PMON@FATAL
sw1(config)# feature nxapi
sw1(config)#
```

Getting back to business

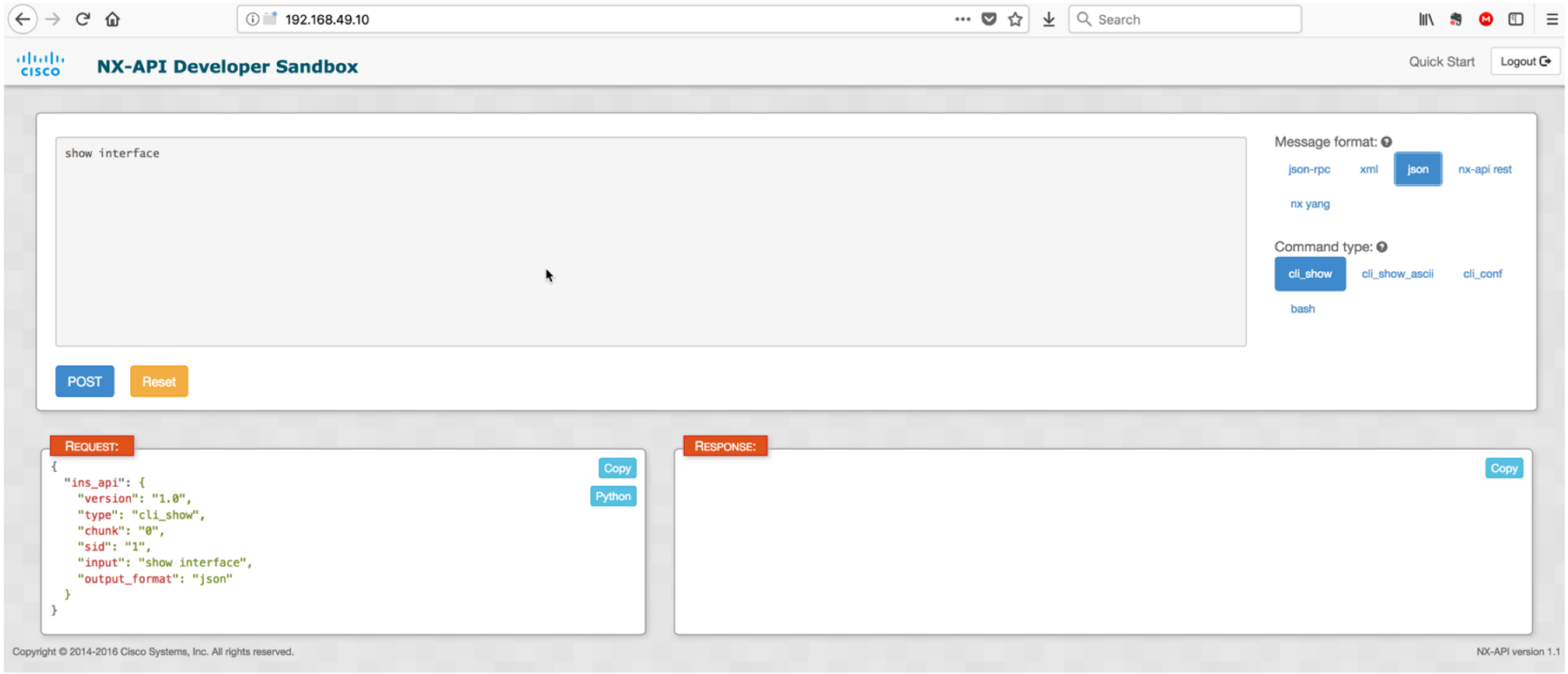
- Setting up our session. Lets create some variables
- request_method = **"POST"**
  url = **"http://192.168.49.10/ins"**
  headers = {**"Content-type"**: **"application/json"**}
  ssl_verify = False
  username = **"admin"**
  password = **"!Passw0rd"**

NX-API works by posting a payload so lets create one

- payload = {

```
    "ins_api": {
        "version": "1.0",
        "type": "cli_show",
        "chunk": "0",
        "sid": "1",
        "input": "show interface ",
        "output_format": "json"
    }
}
```

Where did that come from?



KOVARUS Integrated IT Expertise

So is that just a variable?

- It's a python dictionary which looks exactly like JSON
- Lets play around with something simple:
- my_sample_dictionary = {**"first_key"** : **"some value"**,
  "second_key": **"another value"**,
  "third_key": **"yet another value"**}

```
>>> my_sample_dictionary = {"first_key" : "some value",
...                         "second_key": "another value",
...                         "third_key": "yet another value"}
>>> print my_sample_dictionary["first_key"]
some value
>>>
```

We can modify dictionaries too

- Just set a new value with the “=“

```
Python 2.7.10 (default, Feb  7 2017, 00:08:15)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> my_sample_dictionary = {"first_key" : "some value",
...                         "second_key": "another value",
...                         "third_key": "yet another value"}
>>> print my_sample_dictionary["first_key"]
some value
>>> my_sample_dictionary["first_key"] = "a new value!"
>>> print my_sample_dictionary["first_key"]
a new value!
>>>
```

And we can use other object

- For example we're assigning a variable to a specific key

```
>>> heres_a_variable = 500
>>> my_sample_dictionary["first_key"] = heres_a_variable
>>> print my_sample_dictionary["first_key"]
500
>>>
```

Looking at the API payload

- The ‘input’ field stores the command we’re sending

REQUEST:

Copy

Python

```
{
  "ins_api": {
    "version": "1.0",
    "type": "cli_show",
    "chunk": "0",
    "sid": "1",
    "input": "show interface",
    "output_format": "json"
  }
}
```

Sending it

- Create a new object and we'll store the results in it

```
response = requests.request(request_method,
                            url=url,
                            headers=headers,
                            verify=ssl_verify,
                            data=json.dumps(payload),
                            auth=(username, password))
```

What comes back?

RESPONSE:

Copy

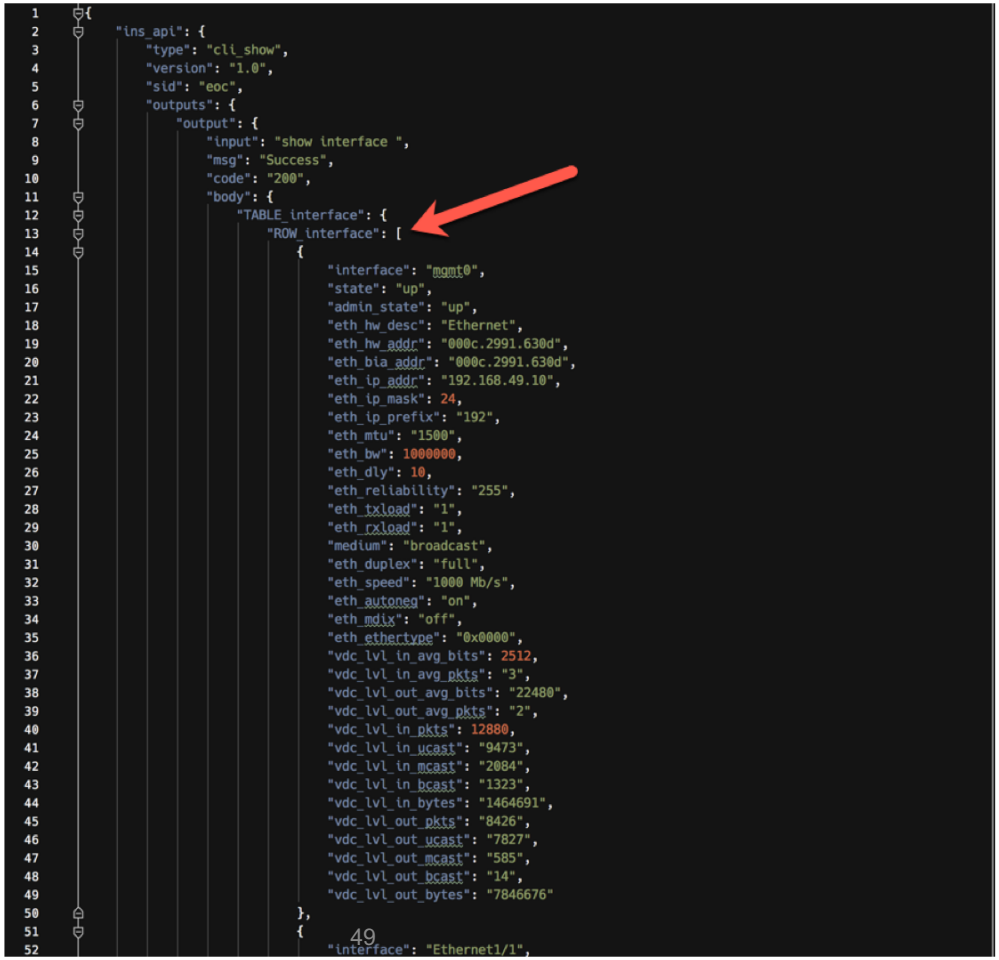

```
{
  "ins_api": {
    "type": "cli_show",
    "version": "1.0",
    "sid": "eoc",
    "outputs": {
      "output": {
        "input": "show interface",
        "msg": "Success",
        "code": "200",
        "body": {
          "TABLE_interface": {
            "ROW_interface": [
              {
                "interface": "mgmt0",
                "state": "up",
                "admin_state": "up",
                "eth_hw_desc": "Ethernet",
                "eth_hw_addr": "000c.2991.630d",
                "eth_bia_addr": "000c.2991.630d",
                "eth_ip_addr": "192.168.49.10",
                "eth_ip_mask": 24,
                "eth_ip_prefix": "192",
                "eth_mtu": "1500",
                "eth_bw": 1000000,
                "eth_dly": 10,
                "eth_reliability": "255",
                "eth_txload": "1",
```


KOVARUS Integrated IT Expertise

Lots of interfaces!

- They’re tucked away deep in the response JSON
- **["ins_api"]["outputs"]["output"]["body"]["TABLE_interface"]["ROW_interface"]**

A list of dictionaries!



KOVARUS
Integrated IT Expertise

We need to iterate through the list to access the interfaces

- Lets create a for loop
- Now each interface gets temporarily stored in a variable called ‘interface’

```
for interface in my_interfaces["ins_api"]["outputs"]["output"]["body"]["TABLE_interface"]["ROW_interface"]:
    print(interface["interface"])
```

Lets grab the state!

- All we need to do is update our print code a little bit:

```
for interface in my_interfaces["ins_api"]["outputs"]["output"]["body"]["TABLE_interface"]["ROW_interface"]:
    print(interface["interface"], interface["state"])
```

Lets run it!

```
Ethernet1/58 down
Ethernet1/59 down
Ethernet1/60 down
Ethernet1/61 down
Ethernet1/62 down
Ethernet1/63 down
Ethernet1/64 down
Traceback (most recent call last):
  File "get_interfaces.py", line 43, in <module>
    print(interface["interface"], interface["state"])
KeyError: 'state'
```

- An error! It seems it doesn't see a key called 'state' in the next interface

Lets have a look at the JSON

- It looks like SVIs have different content in the return JSON


```
{
  "interface": "Ethernet1/64",
  "state": "down",
  "state_rsn_desc": "Link not connected",
  "admin_state": "up",
  "share_state": "Dedicated",
  "eth_hw_desc": "100/1000/10000 Ethernet",
  "eth_hw_addr": "000c.2991.6354",
  "eth_bia_addr": "000c.2991.6354",
  "eth_mtu": "1500",
  "eth_bw": 10000000,
  "eth_dly": 10,
  "eth_reliability": "255",
  "eth_txload": "1",
  "eth_rxload": "1",
  "medium": "broadcast",
  "eth_mode": "access",
  "eth_duplex": "auto",
  "eth_speed": "auto-speed",
  "eth_beacon": "off",
  "eth_autoneg": "on",
```

```
{
  "interface": "Vlan1",
  "svi_admin_state": "down",
  "svi_rsn_desc": "Administratively down",
  "svi_line_proto": "down",
  "svi_mac": " 000c.2991.6314",
  "svi_mtu": "1500",
  "svi_bw": "1000000",
  "svi_delay": "10",
  "svi_tx_load": 1,
  "svi_rx_load": "1",
  "svi_arp_type": "ARPA",
  "svi_time_last_cleared": "never",
  "svi_ucast_pkts_in": "0",
  "svi_ucast_bytes_in": "0"
},
```

Lets fix it!

- We just need to check for the right key. We know ‘state’ is good for switch interfaces and it looks like SVIs use ‘svi_line_proto’

```
for interface in my_interfaces["ins_api"]["outputs"]["output"]["body"]["TABLE_interface"]["ROW_interface"]:
    if "state" in interface.keys():
        print(interface["interface"], interface["state"])
    if "svi_line_proto" in interface.keys():
        print(interface["interface"], interface["svi_line_proto"])
    else:
        pass
```

Lets Create something Fancy

- Lets collect all of the interfaces
- We'll look for any "up" interfaces
- If they're "up" and don't have a description lets output them

Our fancy project

- IP Routes

Some Followup

- You can take arguments with ‘argparse’ which helps re-use
- You can securely prompt for a password with the ‘password’ module
- Learn git!
- https://www.youtube.com/watch?v=HVsySz-h9r4

Fancy Tools

- https://brew.sh/ package management for Macs
- https://www.ansible.com/ automation tool
- https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world an awesome flask tutorial

Good reads (book list!)

- Learn Python the Hard Way
- Automate the Boring Stuff With Python


KOVARUS
Integrated IT Expertise
Appendix