
arista-python-web2py Documentation

Release 0.0.1

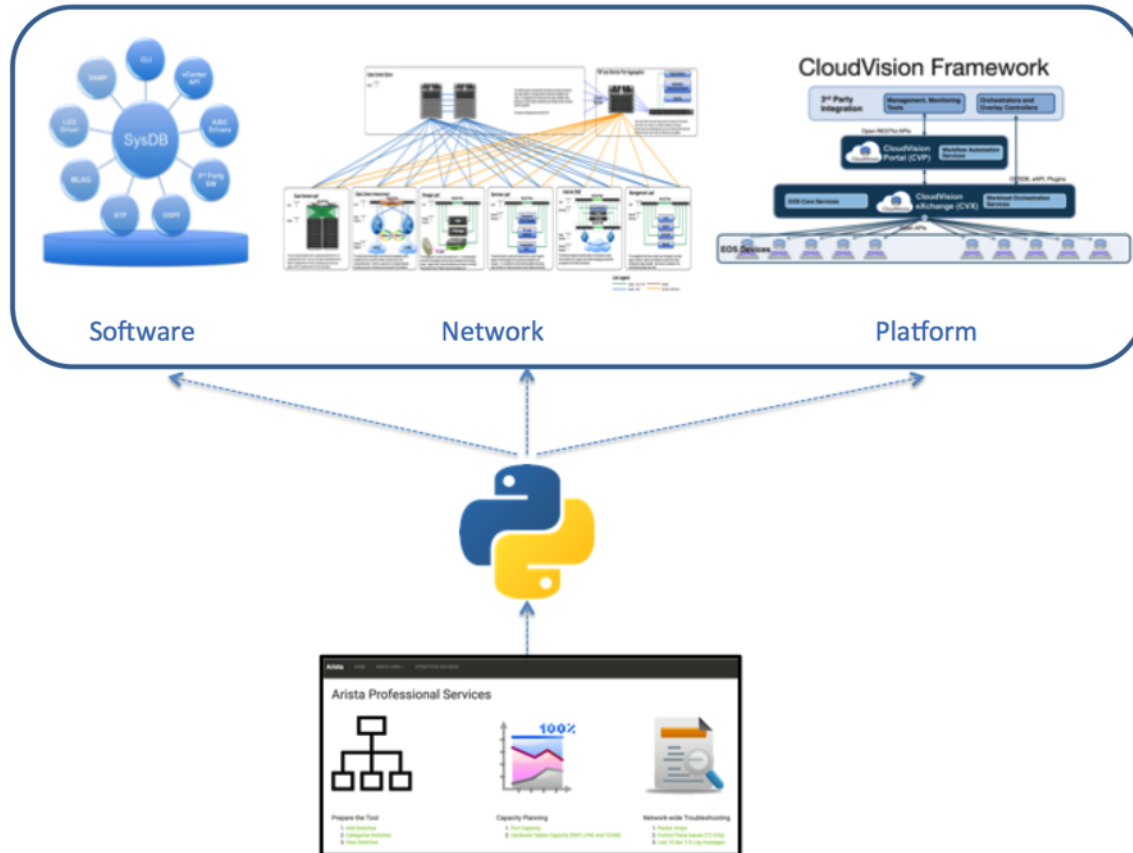
Anees Mohamed

Sep 13, 2017

Contents

1	Preface	3
2	Chapter 1: Introducing Python Core Concepts	5
2.1	Python Implementation on Various Operating Systems	6
2.2	Python Package Manager (pip)	9
2.3	Installing Python Modules	11
2.4	Python Core Concepts	12
2.5	Summary	35
3	Chapter 2: Script Framework for Use Cases	37
3.1	First Script - Show version	37
3.2	Second Script - Running Show Version on Multiple Switches	40
3.3	Third Script - Using External Input	43
3.4	Fourth Script – Storing Result in a Dictionary	49
3.5	Fifth Script – Handling Exceptions	51
3.6	Summary – Script Framework	56
4	Chapter 3: Troubleshooting Use Cases	59
4.1	Monitor Data Plane Drops	59
4.2	Monitor Control Plane Drops	77
5	Chapter 4: Capacity Planning Use Cases	85
5.1	Port Capacity	86
5.2	Hardware Scalability Assessment	98
6	Chapter 5: Web2Py Introduction	125
6.1	Understanding the functionality of the Tool	126
6.2	Web2Py Framework for the Tool	132
7	Chapter 6: Web2Py Installation	137
7.1	Install Required Packages	137
7.2	Install Web2Py	139
7.3	Start Web2Py Service	140
8	Chapter 7: Creating a New Web2Py Application	143
8.1	Default Admin Page	143
8.2	Create a New Application	144

8.3	Edit the Application	147
9	Chapter 8: Web2Py - Prepare the Tool	153
9.1	Add Switches	154
9.2	View Switches	164
9.3	Categorize Switches	165
9.4	Summary	171
10	Chapter 9: Web2Py - Troubleshooting Use Cases	173
10.1	Data Plane Packet Drops	174
10.2	Control Plane Drops	190
10.3	Last 10 Sev 1-3 Log Messages	192
11	Chapter 10: Web2Py - Capacity Planning Use Cases	199
11.1	Port Capacity	200
11.2	Hardware Scale	204
11.3	Summary	206



Contents:

CHAPTER 1

Preface

This book is for network engineers managing Arista network products and have no prior knowledge on any programming language. Python is an easy to learn programming language and more importantly it is a very powerful programming language. Python can be used to build programs ranging from automating simple tasks to building most sophisticated and advanced software applications .

So How does anyone can learn Python?. Typical way of learning programming language is understanding the core concepts with general examples. Then develop programs for the use cases specific to you using the core concepts. So the learning cycle is high because you first learn Python with the examples that you don't need and then translate that learning to create programs for your use case. This is where many of the network engineers lose their learning track.

This book addresses this problem by teaching core Python using the example the network engineers need. The goal of this book is to help network engineers to automate the common operational and troubleshooting steps. Also understanding the programmability of network devices changes your approach to solve a problem in a creative way.

Chapters 1 to 4 discusses Python core concepts, Json RPC and Pyeapi modules with the common networking use cases such as inventory, troubleshooting and capacity planning. You will be introduced with a framework which you can reuse it for your own network use cases.

Chapters 5 to 7 introduces a Python web framework called Web2Py. Chapter 8 to 11 enables you to host your Python programs in the Web2Py application. If you follow Chapters 1 to 11 in this book, you will build and host this web based tool which can be leveraged by all the network engineers in your organization.

Arista

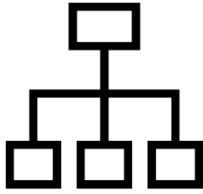
HOME

ARISTA LINKS ▾

OPERATIONS RUN BOOK

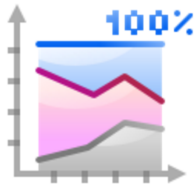
LOG IN ▾

Arista Professional Services




Prepare the Tool

- 1. Add Switches
- 2. Categorize Switches
- 3. View Switches



Capacity Planning

- 1. Port Capacity
- 2. Hardware Tables Capacity (EMT, LPM, and TCAM)



Network-wide Troubleshooting

- 1. Packet Drops
- 2. Control Plane Issues (T2 Only)
- 3. Last 10 Sev 1-3 Log messages

Copyright © 2016

Powered by web2py

4

Chapter 1. Preface

Chapter 1: Introducing Python Core Concepts

- *Python Implementation on Various Operating Systems*
 - *Microsoft Windows*
 - *Apple Mac OS X*
 - *Ubuntu*
- *Python Package Manager (pip)*
- *Installing Python Modules*
 - *Pyeapi*
 - *jsonrpc*
- *Python Core Concepts*
 - *Data Type*
 - *Data Structure*
 - * *List*
 - * *Set*
 - * *Dictionary*
 - *Control Flow Statements*
 - * *if*
 - * *for*
 - *Data Operations*
 - * *String*
 - * *Integer*

- *Simple Use Cases*
- *Script File*
- *Modules*
 - * *pprint*
 - * *sys*
 - * *re - Regular Expression*
- *Handling Structured Data*
 - * *Text*
 - * *JSON*
 - * *YAML*
- *Summary*

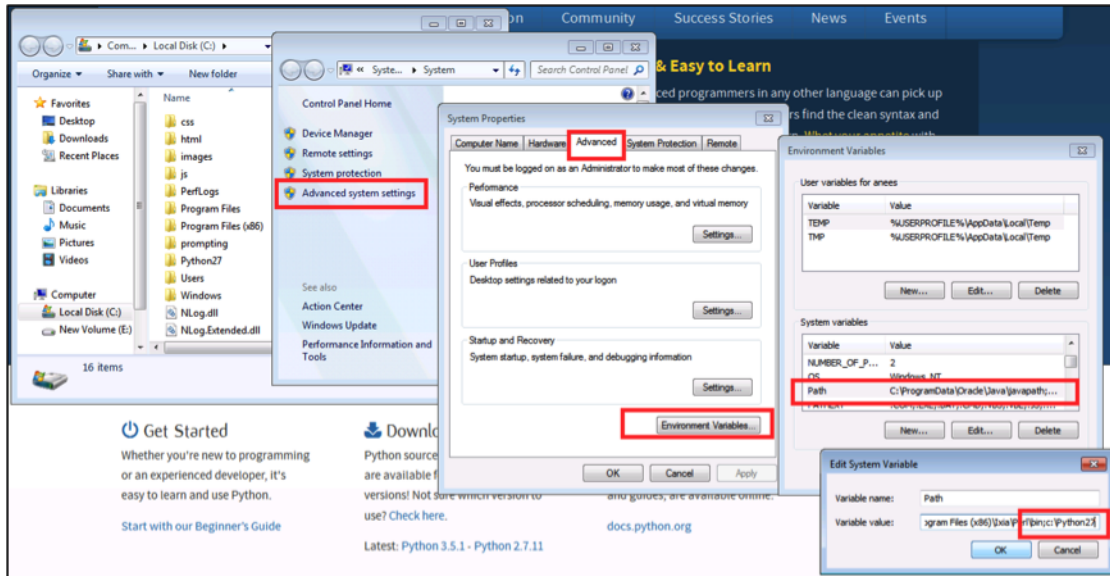
Network engineers require to connect to network devices, collect data using network OS commands and parse through the data to discover the required information. With that in mind, Python's core concepts such as data types, operations, data structures, control flow statements and modules are discussed in this chapter. Before discussing the core concepts, Python implementation on various operating systems need to be discussed.

Python Implementation on Various Operating Systems

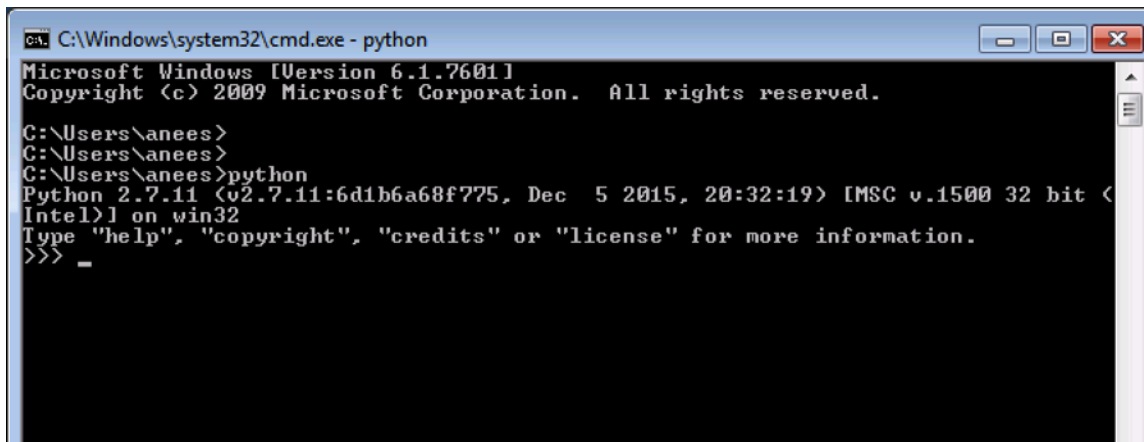
Python implementation on Microsoft Windows, Apple Mac's OS X, Ubuntu Linux distribution and Arista Extensible Operation System (EOS) is discussed in this section. There are two tracks of Python versions available today 2.x and 3.x. Python version 2.x is widely deployed and the industry will eventually move to 3.x. Python programs in this book are written using the version 2.7.x.

Microsoft Windows

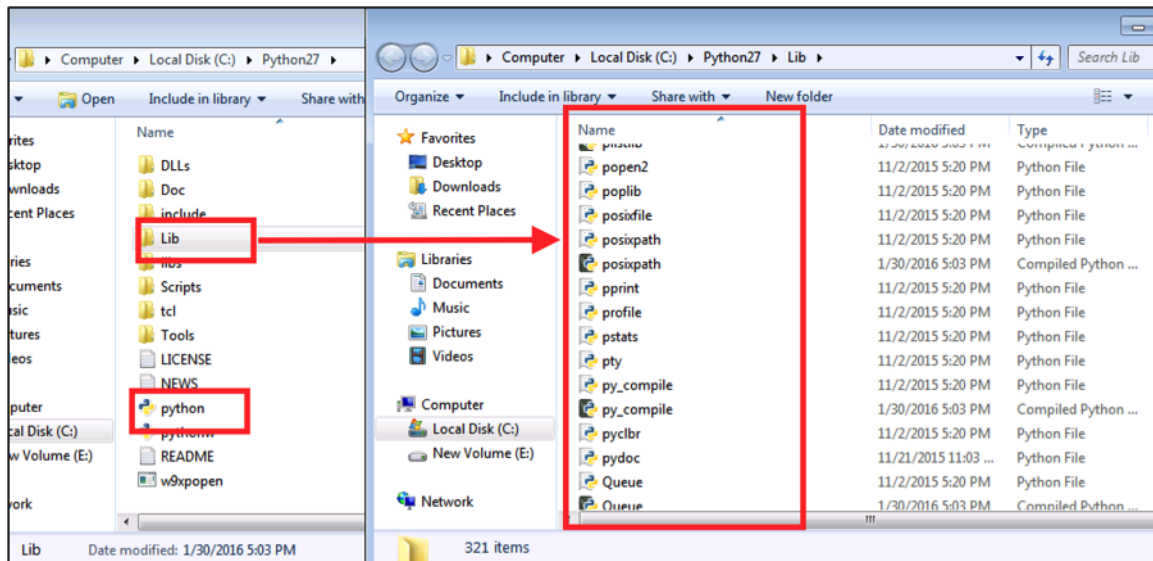
Python is not installed by default on Microsoft Windows operating systems. You can download and install python from www.python.org website. For this book, It is recommended to download and install Python version 2.7.x. By default, Python will be installed in the c:\Python27 folder. After installing Python, PATH variable needs to be updated with the path to the Python installation folder.



After changing the PATH variable, launch the Windows command prompt and verify you can invoke the Python interpreter.



Python interpreter is located in the folder C:\Python27. Python standard library modules are located in C:\Python27\lib folder.



Apple Mac OS X

Apple Mac OS X comes with Python 2.7.x. You can verify by invoking the interpreter from the terminal. If your OS X version does not have Python installed by default, you can download and install python from www.python.org website.

```
anees:~ anees$ python
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

Python interpreter is located in /usr/bin folder and Python standard library files_
↳ are located in /usr/lib/python2.7 folder.

::

anees:~ anees$ which python
/usr/bin/python
anees:~ anees$ ls -l /usr/lib/python2.7
lrwxr-xr-x 1 root wheel 75 Sep 17 05:27 /usr/lib/python2.7 -> ../../System/Library/
↳ Frameworks/Python.framework/Versions/2.7/lib/python2.7

anees:~ anees$ ls -l /usr/lib/python2.7/

***** Skipped *****
-rw-r--r-- 1 root wheel 8252 Aug 22 20:37 posixfile.pyo
-rw-r--r-- 1 root wheel 13925 Aug 22 20:36 posixpath.py
-rw-r--r-- 1 root wheel 12570 Aug 22 20:37 posixpath.pyc
-rw-r--r-- 1 root wheel 12570 Aug 22 20:37 posixpath.pyo
-rw-r--r-- 1 root wheel 11777 Aug 22 20:36 pprint.py
-rw-r--r-- 1 root wheel 11144 Aug 22 20:37 pprint.pyc
-rw-r--r-- 1 root wheel 10967 Aug 22 20:37 pprint.pyo
-rwxr-xr-x 1 root wheel 22782 Aug 22 20:36 profile.py
-rw-r--r-- 1 root wheel 18408 Aug 22 20:37 profile.pyc
-rw-r--r-- 1 root wheel 18161 Aug 22 20:37 profile.pyo
-rw-r--r-- 1 root wheel 26711 Aug 22 20:36 pstats.py
-rw-r--r-- 1 root wheel 28013 Aug 22 20:37 pstats.pyc
```



```
-rw-r--r--      1 root  wheel   28013 Aug 22 20:37 pstats.pyo
***** Skipped *****
```

Ubuntu

Linux distributions come with default Python 2.7.x. Some of the newer distributions come with Python 3.x. Most of the linux software modules are built on top of this default Python version. So upgrading the default Python version on Linux will break those modules. It is recommended to install the desired version using Python virtua environment. For the use cases in this book, the default Python version should be sufficient.

```
anees@ubuntu-web2py:~$ which python
/usr/bin/python
anees@ubuntu-web2py:~$
anees@ubuntu-web2py:~$ python --version
Python 2.7.6

anees@ubuntu-web2py:~$ ls -l /usr/lib/python2.7
total 8228
-rw-r--r--  1 root root  17876 Jun 22  2015 _abcoll.py
-rw-r--r--  1 root root  24794 Dec 29 12:08 _abcoll.pyc
-rw-r--r--  1 root root   7145 Jun 22  2015 abc.py
-rw-r--r--  1 root root   6121 Dec 29 12:08 abc.pyc
-rw-r--r--  1 root root  34231 Jun 22  2015 aifc.py
-rw-r--r--  1 root root  30307 Dec 29 12:08 aifc.pyc
-rw-r--r--  1 root root    60 Jun 22  2015 antigravity.py
-rw-r--r--  1 root root   201 Dec 29 12:08 antigravity.pyc
-rw-r--r--  1 root root   2663 Jun 22  2015 anydbm.py
-rw-r--r--  1 root root   2794 Dec 29 12:08 anydbm.pyc
-rw-r--r--  1 root root   217 Jun 22  2015 argparse.egg-info
-rw-r--r--  1 root root  88691 Jun 22  2015 argparse.py
-rw-r--r--  1 root root  63859 Dec 29 12:08 argparse.pyc
-rw-r--r--  1 root root  11805 Jun 22  2015 ast.py
-rw-r--r--  1 root root  12906 Dec 29 12:08 ast.pyc

***** Skipped *****
```

Python Package Manager (pip)

When Python is installed from the source, it has come with library of standard modules. When a Python interpreter is invoked, very limited number of modules were imported by default into your program's main namespace. Other standard modules in the library can be imported into your program when you need them. There are many vendors, developer community create Python modules and deliver them through pip. pip is a Python package installer which is used to install Python packages from a repository called PyPI (Python Package Index). If you download Python versions 2.7.9 (or 3.4) and above from www.python.org, pip installer is installed by default.

If pip is not installed on your Windows or Mac OS X, you can download the Python program [get-pip.py](#) and install it on your system.

Listing 2.1: Microsoft Windows or Mac OS X

```
python get-pip.py
```

You can verify pip installation on your Windows as described below.

Listing 2.2: Microsoft Windows

```
C:\Users\anees>python -m pip show
ERROR: Please provide a package name or names.
You are using pip version 7.1.2, however version 8.0.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\anees>python -m pip install --upgrade pip
Collecting pip
  Downloading pip-8.0.2-py2.py3-none-any.whl (1.2MB)
    100% |#####| 1.2MB 435kB/s
Installing collected packages: pip
  Found existing installation: pip 7.1.2
  Uninstalling pip-7.1.2:
    Successfully uninstalled pip-7.1.2
  Successfully installed pip-8.0.2
```

You can verify pip installation on your Mac OS X as described below.

Listing 2.3: Apple Mac OS X

```
anees:~ anees$ pip show
ERROR: Please provide a package name or names.
You are using pip version 7.1.2, however version 8.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.

anees:~ anees$ sudo pip install --upgrade pip
Password:
The directory '/Users/anees/Library/Caches/pip/http' or its parent directory is not
↳ owned by the current user and the cache has been disabled. Please check the
↳ permissions and owner of that directory. If executing pip with sudo, you may want
↳ sudo's -H flag.
The directory '/Users/anees/Library/Caches/pip' or its parent directory is not owned
↳ by the current user and caching wheels has been disabled. check the permissions and
↳ owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
Collecting pip
  Downloading pip-8.0.2-py2.py3-none-any.whl (1.2MB)
    100% || 1.2MB 477kB/s
Installing collected packages: pip
  Found existing installation: pip 7.1.2
  Uninstalling pip-7.1.2:
    Successfully uninstalled pip-7.1.2
  Successfully installed pip-8.0.2
anees:~ anees$
```

Since the default Python version on Ubuntu Linux distribution may be prior to 2.7.9, you need to install pip from Linux package manager.

Listing 2.4: Ubuntu

```

anees@ubuntu-web2py:~$ python --version
Python 2.7.6

anees@ubuntu-web2py:~$ sudo apt-get install python-pip

anees@ubuntu-web2py:~$ pip -V
pip 1.5.4 from /usr/lib/python2.7/dist-packages (python 2.7)

```

Later in this book, there are few Python packages installed using pip as and when needed by the use cases. Some of the packages may not be delivered through pip and you can download through your system packet manager or manually download it to the library. For more information to learn about pip, visit <https://pip.pypa.io/en/stable/>.

Installing Python Modules

As mentioned before, there are many vendor and community developed modules available and can be installed using pip. In this book, we primarily use Arista's pyeapi module. We will also use jsonrpc module in some of the use cases in this book. In this section, we will install those modules using pip. If you have not installed pip, refer *Python Package Manager (pip)*.

Pyeapi

```

[admin@ubuntu ~]$ sudo pip install pyeapi
Downloading/unpacking pyeapi
  Downloading pyeapi-0.6.1.tar.gz (115kB): 115kB downloaded
  Running setup.py (path:/tmp/pip_build_root/pyeapi/setup.py) egg_info for package_
↳pyeapi

Downloading/unpacking netaddr (from pyeapi)
  Downloading netaddr-0.7.18-py2.py3-none-any.whl (1.5MB): 1.5MB downloaded
Installing collected packages: pyeapi, netaddr
  Running setup.py install for pyeapi

Successfully installed pyeapi netaddr
Cleaning up...

```

jsonrpc

```

anees:~ anees$ sudo pip install jsonrpc
Collecting jsonrpc
  Downloading jsonrpc-1.2.tar.gz
Installing collected packages: jsonrpc
  Running setup.py install for jsonrpc ... done
Successfully installed jsonrpc-1.2

```

Python Core Concepts

In this section, we are going to discuss the core Python concepts by using simple network use cases. There are six Python core concepts discussed and it is strongly recommended to practice these concepts as you read. You will be surprised how much you can achieve by using these simple concepts as you read through this book.

1. Data Type
2. Data Structure
3. Control Flow Statements
4. Data Operations
5. Modules
6. Handling Structured Data

The approach of this book is to learn by practice. This is one of the reason we chose to teach Python using networking use cases. When you practice using the use cases you know, we don't necessarily explain every concepts textually. We recommend you to practice these concepts multiple times. You also don't restrict yourselves to the use cases discussed in this book. Expand the practice with your own networking use cases.

Data Type

A couple of data types important to us is strings and integers. Launch the Python interpreter from your terminal.

```
anees:~ anees$ python
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Below are the examples for three data formats string, integer and floating point.

```
>>> switch = "10.10.10.11"
>>> type(switch)
<type 'str'>

>>> last_octet = 11
>>> type(last_octet)
<type 'int'>

>>> temperature = 39.2
>>> type(temperature)
<type 'float'>
```

The line `switch = "10.10.10.11"` is called as **assignment statement**. The “switch” is called as **variable** and the “10.10.10.11” is called as **value**. Observe the spaces between variables and values. Refer [Python Style Guide for whitespace in expressions](#) for more information.

Strings are represented within quotes. There are multiple ways to represent strings. Below are the examples of representing strings using single, double and triple quotes. If you need to type the text in multiple lines as you see in the variable `switch3`, you have to use triple quotes.

```
>>> switch1 = 'description "CORE" switch'
>>>
>>> switch2 = "interface ethernet1"
```

```
>>>
>>> switch3 = """interface ethernet1
... ip address 10.10.20.2/24
... no shutdown
... """
>>>
```

You can view the content of the variable directly typing the variable name.

```
>>> switch3
'interface ethernet1\n ip address 10.10.20.2/24\n no shutdown\n'
```

“\n” is an escape character in Python to represent new line. If you need to type the multi line text in single line as you see above, you have to use escape characters within single or double quotes.

When you print the content of this string using “print” module, the data is presented in readable format instead of displaying in the string internal format using escape characters.

```
>>> print switch3
interface ethernet1
 ip address 10.10.20.2/24
 no shutdown
```

When the data is stored in the system, it has to be encoded in a specific format. Unicode is an industry standard encoding format. We will be often converting unicode to string when we display the data to the end user.

```
>>> switch4 = u"router bgp 100"
>>>
>>> switch4
u'router bgp 100'
>>>
>>> str(switch4)
'router bgp 100'
>>>
>>> switch5 = str(switch4)
>>> switch5
'router bgp 100'
```

Data Structure

Data structure is a way of organizing the data in a system. String can be considered as a data structure as well. Data structure helps us to access the data efficiently. We are going to see three Python data structures in this section. Those are list, set and dictionary. We will use list and dictionary extensively throughout this book.

List

List is a Python data structure to store a list of values. List is represented using comma-separated values inside a square [] bracket.

```
>>> device_list = ["10.10.10.13", "10.10.10.11", "10.10.10.14", "10.10.10.12"]
>>>
>>> type(device_list)
<type 'list'>
>>>
```

```
>>> device_list
['10.10.10.13', '10.10.10.11', '10.10.10.14', '10.10.10.12']
```

How do you access a specific value in the list? You can access the value by using the index within the square bracket.

```
>>> device_list[0]
'10.10.10.13'
>>> device_list[1]
'10.10.10.11'
>>> device_list[3]
'10.10.10.12'
```

How can we modify the list? You can add the items using `append()` method and you can update the existing item using assignment statement.

```
>>> device_list
['10.10.10.11', '10.10.10.12', '10.10.10.13']
>>>
>>> device_list.append("10.10.10.12")
>>> device_list
['10.10.10.11', '10.10.10.12', '10.10.10.13', '10.10.10.12']
>>>
>>> new_ip = "10.10.10.15"
>>> device_list.append(new_ip)
>>> device_list
['10.10.10.11', '10.10.10.12', '10.10.10.13', '10.10.10.12', '10.10.10.15']
>>>
>>> device_list[0] = new_ip
>>> device_list
['10.10.10.15', '10.10.10.12', '10.10.10.13', '10.10.10.12', '10.10.10.15']
```

As you can see, list can have duplicate values. Where can I find the list of operations (methods) that can be performed on the list?

```
>>> dir(device_list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__delslice__',
→ '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__
→ getslice__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__',
→ '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__
→ reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
→ '__setslice__', '__sizeof__', '__str__', '__subclasshook__',
'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

>>> help(device_list)
|
| append(...)
|     L.append(object) -- append object to end
|
| count(...)
|     L.count(value) -> integer -- return number of occurrences of value
|
| extend(...)
|     L.extend(iterable) -- extend list by appending elements from the iterable
|
| index(...)
|     L.index(value, [start, [stop]]) -> integer -- return first index of value.
|     Raises ValueError if the value is not present.
|
```

```

| insert(...)
|     L.insert(index, object) -- insert object before index
|
| pop(...)
|     L.pop([index]) -> item -- remove and return item at index (default last).
|     Raises IndexError if list is empty or index is out of range.
|
| remove(...)
|     L.remove(value) -- remove first occurrence of value.
|     Raises ValueError if the value is not present.
|
| reverse(...)
|     L.reverse() -- reverse *IN PLACE*
|
| sort(...)
|     L.sort(cmp=None, key=None, reverse=False) -- stable sort *IN PLACE*;
|     cmp(x, y) -> -1, 0, 1
:
<Type q to exit>

```

Let us explore few more methods over the list.

```

>>> device_list = ['10.10.10.11', '10.10.10.12', '10.10.10.13', '10.10.10.14', '10.10.
↪10.15']

>>> device_list.reverse()
>>> device_list
['10.10.10.15', '10.10.10.14', '10.10.10.13', '10.10.10.12', '10.10.10.11']

>>> device_list.sort()
>>> device_list
['10.10.10.11', '10.10.10.12', '10.10.10.13', '10.10.10.14', '10.10.10.15']

>>> device_list.pop()
'10.10.10.15'
>>> device_list
['10.10.10.11', '10.10.10.12', '10.10.10.13', '10.10.10.14']

>>> device_list.remove("10.10.10.12")
>>> device_list
['10.10.10.11', '10.10.10.13', '10.10.10.14']

```

Set

Set is similar to list with few differences:

- Items in set are unique. It eliminates duplicate entries.
- Unordered list of items.
- Supports powerful operations such as difference, union and intersection between sets.

Let us create a set and practice some of the basic operations related to set.

```

>>> device_set = set(["10.10.10.11", "10.10.10.12"])
>>>
>>> device_set
set(['10.10.10.11', '10.10.10.12'])

```

```
>>> type(device_set)
<type 'set'>

>>> device_set.add("10.10.10.13")
>>> device_set
set(['10.10.10.11', '10.10.10.13', '10.10.10.12'])

>>> device_set.remove("10.10.10.12")
>>> device_set
set(['10.10.10.11', '10.10.10.13'])

>>> new_ip = "10.10.10.13"
>>> device_set.add(new_ip)
>>> device_set
set(['10.10.10.11', '10.10.10.13'])
```

You can convert a list to set. If the list has duplicate entries, converting to set will eliminate the duplicate entries.

```
>>> device_list
['10.10.10.15', '10.10.10.12', '10.10.10.13', '10.10.10.12', '10.10.10.15']

>>> device_list_to_set = set(device_list)
>>> device_list_to_set
set(['10.10.10.15', '10.10.10.13', '10.10.10.12'])
```

Let us explore the methods specific to set.

```
>>> dir(device_set)
['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__', '__eq__',
→ '__format__', '__ge__', '__getattr__', '__gt__', '__hash__', '__iand__', '__
→ 'init__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__
→ ', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__
→ repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__',
→ '__sub__', '__subclasshook__', '__xor__',
'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection',
→ 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove',
→ 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']

>>> switch1_neighbors = set(["switch2", "switch3", "switch4"])
>>> switch2_neighbors = set(["switch1", "switch3", "switch5"])

>>> switch1_neighbors.intersection(switch2_neighbors)
set(['switch3'])

>>> switch1_neighbors.union(switch2_neighbors)
set(['switch3', 'switch2', 'switch1', 'switch5', 'switch4'])

>>> switch2_neighbors.difference(switch1_neighbors)
set(['switch1', 'switch5'])
```

Dictionary

Dictionary is a list of {key: value} items and defined using curly brackets. Let us look at the basic operations using dictionary.


```
>>> inventory = {"10.10.10.11": "spine-1", "10.10.10.12": "spine-2"}
>>> type(inventory)
<type 'dict'>

>>> inventory["10.10.10.11"]
'spine-1'

>>> inventory["10.10.10.13"] = "tor-1"
>>> inventory
{'10.10.10.11': 'spine-1', '10.10.10.13': 'tor-1', '10.10.10.12': 'spine-2'}

>>> inventory.keys()
['10.10.10.11', '10.10.10.13', '10.10.10.12']
```

The values of the dictionary can be a simple string, a list or another dictionary. Below is an example that shows a value of a key which is another dictionary.

```
>>> inventory["10.10.10.11"] = {"hostname": "spine-1", "version": "4.15.4F", "model":
↳ "7050SX-128"}
>>> inventory
{'10.10.10.11': {'model': '7050SX-128', 'hostname': 'spine-1', 'version': '4.15.4F'},
↳ '10.10.10.13': 'tor-1-a', '10.10.10.12': 'spine-2'}

>>> inventory["10.10.10.11"]
{'model': '7050SX-128', 'hostname': 'spine-1', 'version': '4.15.4F'}

>>> inventory["10.10.10.11"]["version"]
'4.15.4F'
```

Control Flow Statements

The flow of the script can be changed by conditional and loop statements. We are going to take a look at “if clause” and “for loop” in this section.

if

Here is an example of using simple if clause. We are also using the operators == (Equal to) and != (Not Equal to).

```
>>> interface = "management1"
>>>
>>> if interface == "management1":
...     print "It is the management interface"
...
It is the management interface

>>> if interface != "management1":
...     print "It is NOT the management interface"
>>>
```

The statements following if statement are indented by 4 spaces. Refer [Python Style Guide for Indentation](#) for more information. Here is an example of using if and else clause. We are using the operator <= (Less than equal to).

```
>>> max_interfaces = 36

>>> n = 10
```

```
>>> if n <= max_interfaces:
...     print "Interface number is within the range"
... else:
...     print "Interface number is not within the range"
...
Interface number is within the range
```

Here is an example of using if, elif and else clause. We are using the operator “not in” against the list.

```
>>> device_list = ["10.10.10.11", "10.10.10.12", "10.10.10.13"]

>>> if "10.10.10.11" not in device_list:
...     print "10.10.10.11 is not in the list"
... elif "10.10.10.12" not in device_list:
...     print "10.10.10.12 is not in the list"
... elif "10.10.10.13" not in device_list:
...     print "10.10.10.13 not in device_list"
... else:
...     print "all the IPs are in the device_list"
...
all the IPs are in the device_list
```

Here is an example to find whether a list is empty or not using if clause.

```
>>> device_list = []
>>> if not device_list:
...     print "Empty List"
...
Empty List

>>> device_list = ["10.10.10.11"]
>>> if not device_list:
...     print "Empty List"
... else:
...     print "Not Empty"
...
Not Empty
```

Here is an example to find whether a dictionary is empty or not using if clause.

```
>>> inventory = {"10.10.10.11": "host1"}

>>> not inventory
False
>>>
>>> bool(inventory)
True

>>> if bool(inventory):
...     print inventory
...
{'10.10.10.11': 'host1'}

>>> if not inventory:
...     print "Empty dictionary"
... else:
...     print inventory
```

```

...
{'10.10.10.11': 'host1'}

*** With Empty Dictionary ***

>>> inventory = {}

>>> not inventory
True
>>> bool(inventory)
False

>>> if not inventory:
...     print "Empty dictionary"
...
Empty dictionary

>>> if bool(inventory):
...     print "NOT EMPTY"
... else:
...     print "Empty Dictionary"
...
Empty Dictionary

```

for

for loop is used to iterate over a sequence of items. Below is an example of iterating list and dictionary.

```

>>> device_list = ["10.10.10.11", "10.10.10.12", "10.10.10.13"]
>>>
>>> for each_ip in device_list:
...     print each_ip
...
10.10.10.11
10.10.10.12
10.10.10.13

>>> inventory = {'10.10.10.11': {'model': '7050SX-128', 'hostname': 'spine-1',
↪ 'version': '4.15.4F'}, '10.10.10.12': {'model': '7050SX-128', 'hostname': 'spine-2
↪ ', 'version': '4.15.4F'}, '10.10.10.13': {'model': '7050SX-128', 'hostname': 'leaf-1
↪ ', 'version': '4.15.6M'}}

>>> for each_ip in inventory:
...     print each_ip
...
10.10.10.11
10.10.10.13
10.10.10.12

>>> for each_ip in inventory:
...     print inventory[each_ip]["hostname"]
...
spine-1
leaf-1
spine-2

```

Data Operations

In this section we are going to discuss about the various in built methods for strings and integers.

String

By now you know how to find the supported methods (`dir()`) for various types of data structures and data types. Let us practice some basic methods on strings.

```
>>> ip_address = "ip address 10.10.10.11/24"
>>> ip_address.split()
['ip', 'address', '10.10.10.11/24']

>>> ip_address.split()[2]
'10.10.10.11/24'

>>> ip_address.split()[2].split("/")
['10.10.10.11', '24']

>>> ip_address.split()[2].split("/")[0]
'10.10.10.11'
```

`split()` splits the string into list. The default delimiter is empty space.

Now let us practice how we can combine strings.

```
>>> ip = "10.10.10.100"
>>> subnet_mask = 24

>>> ip_addr = "ip address" + ip
>>> ip_addr
'ip address10.10.10.100'

>>> ip_addr = "ip address " + ip
>>> ip_addr
'ip address 10.10.10.100'

>>> ip_statement = ip_addr + "/" + subnet_mask
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects

>>> type(subnet_mask)
<type 'int'>

>>> ip_statement = ip_addr + "/" + str(subnet_mask)
>>> ip_statement
'ip address 10.10.10.100/24'
```

As you see when we try to add a string and integer, Python reports **TypeError: cannot concatenate 'str' and 'int' objects**. So we converted the integer to string using `str(subnet_mask)` method and concatenated to `ip_addr` string variable.

Integer

Here are the some of the examples of methods over integers and floating point.

```

>>> number_of_line_cards = 8
>>> ports_per_line_card = 36

>>> total_number_of_ports = number_of_line_cards * ports_per_line_card
>>> total_number_of_ports
288

>>> used_ports = 120
>>> free_ports = total_number_of_ports - used_ports
>>> free_ports
168

>>> ports_usage_in_percentage = (used_ports/total_number_of_ports) * 100
>>> ports_usage_in_percentage
0

>>> 120/288
0
>>> float(120/288)
0.0
>>> 120.0/288.0
0.4166666666666667

>>> ports_usage_in_percentage = (float(used_ports)/float(total_number_of_ports) * 100)
>>> ports_usage_in_percentage
41.66666666666667

```

“free_ports = total_number_of_ports - used_ports” is called as **statement**. In that, “total_number_of_ports - used_ports” is called as **expression**. Within the expression, total_number_of_ports, used_ports are variables and “-” is called as operator.

Simple Use Cases

We will practice few simple use cases based on the concepts we have learned so far. Here is an example of using for loop and string methods. In this example, we will extract the IP addresses from the “show ip interface brief” output.

```

>>> ip_int_br = """ Interface           IP Address      Status      Protocol      MTU
↳ Ethernet1/1.101      192.168.121.0/31 up           up           1500
↳ Ethernet2/1.101      192.168.221.0/31 down        lowerlayerdown 1500
↳ Ethernet3/1.301      10.12.1.0/31    up           up           1500
↳ Ethernet3/1.317      10.12.1.32/31   up           up           1500
↳ Ethernet3/1.333      10.12.1.64/31   up           up           1500
↳ Ethernet3/1.349      10.12.1.96/31   up           up           1500 """

>>> ip_int_br
' Interface           IP Address      Status      Protocol      MTU'
↳ MTU\nEthernet1/1.101      192.168.121.0/31 up           up           1500'
↳ 1500\nEthernet2/1.101      192.168.221.0/31 down        lowerlayerdown 1500'
↳ 1500\nEthernet3/1.301      10.12.1.0/31    up           up           1500'
↳ 1500\nEthernet3/1.317      10.12.1.32/31   up           up           1500'
↳ 1500\nEthernet3/1.333      10.12.1.64/31   up           up           1500'
↳ 1500\nEthernet3/1.349      10.12.1.96/31   up           up           1500'

>>> ip_int_br.split("\n")
[' Interface           IP Address      Status      Protocol      MTU',
↳ 'Ethernet1/1.101      192.168.121.0/31 up           up           1500',
↳ 'Ethernet2/1.101      192.168.221.0/31 down        lowerlayerdown 1500',
↳ 'Ethernet3/1.301      10.12.1.0/31    up           up           1500',
↳ 'Ethernet3/1.317      10.12.1.32/31   up           up           1500',
↳ 'Ethernet3/1.333      10.12.1.64/31   up           up           1500',
↳ 'Ethernet3/1.349      10.12.1.96/31   up           up           1500']

```

```
>>> for each_line in ip_int_br.split("\n"):
...     print each_line
...
Interface          IP Address      Status    Protocol      MTU
Ethernet1/1.101    192.168.121.0/31 up        up            1500
Ethernet2/1.101    192.168.221.0/31 down      lowerlayerdown 1500
Ethernet3/1.301    10.12.1.0/31    up        up            1500
Ethernet3/1.317    10.12.1.32/31   up        up            1500
Ethernet3/1.333    10.12.1.64/31   up        up            1500
Ethernet3/1.349    10.12.1.96/31   up        up            1500

>>> for each_line in ip_int_br.split("\n"):
...     print each_line.split()[1]
...
IP
192.168.121.0/31
192.168.221.0/31
10.12.1.0/31
10.12.1.32/31
10.12.1.64/31
10.12.1.96/31
```

Here is another example of using for loop, string and integer methods. We will calculate the number of subnets and IP addresses (including network and broadcast IP addresses) from the given network address and subnet mask.

```
>>> network_block = "192.168.1.0/24"
>>> subnet_mask = 30

*** Identify the number of bits used for network. "subnet_mask - network_mask" *****
>>> network_block.split("/")
['192.168.1.0', '24']
>>> network_block.split("/") [1]
'24'
>>> subnet_mask - int(network_block.split("/") [1])
6

*** Number of subnets = 2 to the power of (subnet_mask - network_mask) ***
>>> number_of_subnets = 2 ** (subnet_mask - int(network_block.split("/") [1]))
>>> number_of_subnets
64

*** Number of IPs per subnet = 2 to the power of number of host bits ***
>>> number_of_ips_per_subnet = 2 ** (32 - subnet_mask)
>>> number_of_ips_per_subnet
4
```

If you look at some of the statements, we have more than one operators in the expression. When you have more than one operators in an expression, Python follows the conventional method called as PEMDAS (Parenthesis, Exponents, Multiplication, Division, Addition, Subtraction) to calculate the result.

Now, we will continue to update our script to calculate the subnet addresses for the given network block and subnet mask.

```
>>> subnet_octets = network_block.split("/") [0].split(".")
>>> subnet_octets
['192', '168', '1', '0']
```

```
>>> ".".join(subnet_octets)
'192.168.1.0'

>>> subnetwork_address = int(subnet_octets[3])
>>> for each_subnet in range(0, number_of_subnets):
...     subnet_octets[3] = str(subnetwork_address)
...     subnets = ".".join(subnet_octets) + "/" + str(subnet_mask)
...     print subnets
...     subnetwork_address += number_of_ips_per_subnet
...
192.168.1.0/30
192.168.1.4/30
192.168.1.8/30
192.168.1.12/30
192.168.1.16/30
!!!! Output Omitted for brevity
192.168.1.228/30
192.168.1.232/30
192.168.1.236/30
192.168.1.240/30
192.168.1.244/30
192.168.1.248/30
192.168.1.252/30
```

Within the for loop, we are just updating the fourth octet of the IP addresses and appending the string to list the subnet addresses.

Script File

So far, We have been practicing the core concepts using Python interpreter. This approach of using interpreter is called as interactive mode. As you see in the previous use case, it is getting complicated to use the interpreter as the script gets complicated. It is time to start writing scripts in a file which will be saved as .py file in your system. Then the script file is executed directly from your terminal or from any development environment. Now, the question comes what editor should we use to create the .py file. One could use a simple text editor to write the scripts. Working with the text editor is something similar to writing script in the Python interpreter where you have to make sure you write the code with correct indentation and correct syntax.

There are sophisticated advanced editors that can provide auto indentation and in some cases auto completing the statements. You can start writing the script using one of the advanced editors such as [Atom](#) or [Sublime](#).

Later in this book, we use Python's Integrated Development Environment (IDLE) to create scripts. IDLE is installed as part of the Python software package when you download and install from www.python.org. There are several vendor developed integrated development environments (IDE) such as Wing IDE, PyCharm, Komodo, etc are available in the market. In addition to the benefits provided by advanced editors, IDEs are great while developing, testing and debugging complex software applications.

Create a folder in the home directory of your system and save the scripts you are writing in that folder. Open your choice of editor and create a new script and save the file as subnet_calculator.py.

We are going to take the script we built so far using interactive mode and paste it in this new script file. The script we have written so far is very specific to class C subnets. We are going to add a logic that automatically derive the class from the given network block and subnet mask.

```
network_block = "10.32.10.0/24"
subnet_mask = 26

# split the network block into IP address and Network Mask
```

```
ip_address = network_block.split("/")[0]
network_mask = int(network_block.split("/")[1])
octets = ip_address.split(".")

# Find the octet number for which subnet address needs to be calculated
# class A = 1, Class B = 2, Class C = 3
subnet_octet = network_mask / 8

# Find out how many subnets for a given network mask and subnet mask
number_of_subnets = 2 ** (subnet_mask - network_mask)

# Total number of IPs within a subnet
# host class for class A, B and C is 16, 24 and 32 respectively.
host_class = (subnet_octet * 8) + 8
number_of_ips_per_subnet = 2 ** (host_class - subnet_mask)

# Derive the subnet addresses
subnetwork_address = int(octets[subnet_octet])

for each_subnet in range(0, number_of_subnets):
    octets[subnet_octet] = str(subnetwork_address)
    subnets = ".".join(octets) + "/" + str(subnet_mask)
    print subnets
    subnetwork_address += number_of_ips_per_subnet
```

It may take sometime to understand the logic of the script. Later in this book, we will learn to write an algorithm and we develop all the scripts step by step by following the algorithm. For now, just save and run the script from your system.

```
anees:my-scripts anees$ ls subnet*
subnet_calculator.py

anees:my-scripts anees$ python subnet_calculator.py
10.32.10.0/26
10.32.10.64/26
10.32.10.128/26
10.32.10.192/26
```

Modules

As you start learning to write scripts, you will end up in reusing some of your programs. You can write the reusable scripts as functions in Python. You can also write all of your reusable use cases as multiple functions and save it in a .py file. Then for any new programs or scripts, you can import these functions in the .py file and use it. This .py file is called as module in Python.

In some cases, you may end up in building multiple modules and you may need all these modules to build more sophisticated software applications. Collection of these modules can be called as packages in Python.

There are several standard modules or packages installed as part of the Python installation. You can find those packages in the lib folder of Python installation in your system. For example in Windows, c:\python2.7\lib and in Linux & MAC OS X, /usr/lib/python2.7/ have the standard Python modules.

We will explore some of the standard modules (pprint, sys and re) in this section. Launch the Python interpreter from your terminal.

pprint

Pretty printer (pprint) is a very useful module especially when we handle dictionaries. In order to use the modules, you must import them into your script.

```
>>> import pprint
>>>
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'pprint']
>>> dir(pprint)
['PrettyPrinter', '_StringIO', '__all__', '__builtins__', '__doc__', '__file__', '__
↳name__', '__package__', '_commajoin', '_id', '_len', '_perfcheck', '_recursion', '_
↳safe_repr', '_sorted', '_sys', '_type',
'isreadable', 'isrecursive', 'pformat', 'pprint', 'saferepr', 'warnings']

>>> inventory = {'10.10.10.11': {'model': '7050SX-128', 'hostname': 'spine-1',
↳'version': '4.15.4F'}, '10.10.10.13': 'tor-1-a', '10.10.10.12': 'spine-2'}

>>> print inventory
{'10.10.10.11': {'model': '7050SX-128', 'hostname': 'spine-1', 'version': '4.15.4F'},
↳'10.10.10.13': 'tor-1-a', '10.10.10.12': 'spine-2'}

>>> pprint.pprint(inventory)
{'10.10.10.11': {'hostname': 'spine-1',
                  'model': '7050SX-128',
                  'version': '4.15.4F'},
 '10.10.10.12': 'spine-2',
 '10.10.10.13': 'tor-1-a'}
```

Both print and pprint are inbuilt modules in Python. But print is loaded in your program namespace when you launch the Python program by default. You can see the difference between the outputs of the dictionary using print and pprint.

sys

sys module provide system specific functions which can be used to interact with the systems (Microsoft Windows, Apple Mac, Linux, etc) objects and variables. In the subnet_calculator.py, we specify the network address and the subnet mask within the script. We are going to use sys module to input both of these values as arguments instead of hard coding into the script.

```
import sys

network_block = sys.argv[1]
subnet_mask = int(sys.argv[2])

# split the network block into IP address and Network Mask
ip_address = network_block.split("/") [0]
network_mask = int(network_block.split("/") [1])
octets = ip_address.split(".")

# Find the octet number for which subnet address needs to be calculated
subnet_octet = network_mask / 8

# Find out how many subnets for a given network mask and subnet mask
number_of_subnets = 2 ** (subnet_mask - network_mask)

# Total number of IPs within a subnet
host_class = (subnet_octet * 8) + 8
```

```
number_of_ips_per_subnet = 2 ** (host_class - subnet_mask)

# Derive the subnet addresses
subnet_network_address = int(octets[subnet_octet])

for each_subnet in range(0, number_of_subnets):
    octets[subnet_octet] = str(subnet_network_address)
    subnets = ".".join(octets) + "/" + str(subnet_mask)
    print subnets
    subnet_network_address += number_of_ips_per_subnet
```

Save and run the script from your system.

```
anees:my-scripts anees$ ls subnet*
subnet_calculator.py

anees:my-scripts anees$ python subnet_calculator.py 10.10.10.0/24 26
10.10.10.0/26
10.10.10.64/26
10.10.10.128/26
10.10.10.192/26
```

What will happen if the user does not provide the arguments?

```
anees:my-scripts anees$ python subnet_calculator.py
Traceback (most recent call last):
  File "subnet_calculator.py", line 3, in <module>
    network_block = sys.argv[1]
IndexError: list index out of range
```

We can add a validation check in the script to make sure the user provides two arguments. If the user does not provide the arguments, the script should display a message that educates the user.

```
import sys

if len( sys.argv ) <= 2:
    sys.stderr.write("Example Syntax \n")
    sys.stderr.write("python subnet_calculator.py 192.168.1.0/24 28 \n")
    sys.exit( 1 )

network_block = sys.argv[1]
subnet_mask = int(sys.argv[2])

# split the network block into IP address and Network Mask
ip_address = network_block.split("/") [0]
network_mask = int(network_block.split("/")[1])
octets = ip_address.split(".")

# Find which octet for which subnet address needs to be calculated
# class A = 1, Class B = 2, Class C = 3
subnet_octet = network_mask / 8

# Find out how many subnets for a given network mask and subnet mask
number_of_subnets = 2 ** (subnet_mask - network_mask)

# Total number of IPs within a subnet
# host class for class A, B and C is 16, 24 and 32.
host_class = (subnet_octet * 8) + 8
```

```

number_of_ips_per_subnet = 2 ** (host_class - subnet_mask)

# Derive the subnet addresses
subnetwork_address = int(octets[subnet_octet])

for each_subnet in range(0, number_of_subnets):
    octets[subnet_octet] = str(subnetwork_address)
    subnets = ".".join(octets) + "/" + str(subnet_mask)
    print subnets
    subnetwork_address += number_of_ips_per_subnet

```

Save and run the script from your system.

```

anees:my-scripts anees$ python subnet_calculator.py
Example Syntax
python subnet_calculator.py 192.168.1.0/24 28

```

re - Regular Expression

Regular expression is a powerful method when it comes to automation. We are going to practice a couple of use cases in this section using regular expression.

Best Practices Assessment

In this first use case, we are going to use `re.search()` method. We are going to practice a script to validate if the bgp best practices commands such as “update wait-for-convergence” and “update wait-install” are configured.

```

>>> import re
>>>
>>> config = """
... interface Loopback0
...     description loopback interface for BGP peering
...     ip address 192.168.100.2/32
... !
... router bgp 100
...     update wait-for-convergence
...     maximum-paths 4
...     neighbor 10.1.1.1 remote-as 1001
...     neighbor 10.1.1.1 maximum-routes 12000
...     neighbor 192.168.128.18 remote-as 201
...     neighbor 192.168.128.18 maximum-routes 12000
...     neighbor 192.168.128.22 remote-as 201
...     neighbor 192.168.128.22 maximum-routes 12000
...     network 10.10.10.0/24
...     network 192.168.100.2/32
... """
>>>
>>> validate = re.search("update wait-for-convergence", config)
>>> validate
<_sre.SRE_Match object at 0x100f5d9f0>

>>> if validate:
...     print "Match Found"
... else:
...     print "Match NOT Found"
...
Match Found

```

```
>>>
>>> validate = re.search("update wait-install", config)
>>> validate
>>>
>>> if validate:
...     print "Match Found"
... else:
...     print "Match NOT Found"
...
Match NOT Found
```

If a match is found, `re.search()` will return a `Match Object` instance. If a match is not found, it returns `None`.

Configuration Generator - Collecting the variables from the configuration template

In this second use case, we are going to use `re.findall()` method. Network engineers can standardize network device configurations. Once they standardize network device configurations, they can create configuration template which can be used to generate configurations for any number of devices by filling up the variables such as hostname, IP addresses, etc. In this example, we will show how we can collect the variables from the configuration template. Later in this chapter, we will complete the script by replacing the variables with the actual values.

```
Import re

>>> config = """
... hostname $hostname
... !
... interface management1
...   ip address $ip_mgmt/24
... !
... interface loopback0
...   ip address $ip_lo0/32
... !
... """

>>> re.findall("\$\w+", config)
['$hostname', '$ip_mgmt', '$ip_lo0']
```

Handling Structured Data

In this section, we are going to complete the configuration generator script which we started in the regular expression section. We are going to save the configuration template in a text file and the variables will be stored in a json or yaml file.

Text

Create a configuration template and store it in a text file.

```
anees:files anees$ pwd
/Users/anees/Dropbox/my-scripts/files

anees:files anees$ vi config_template.txt
hostname $hostname
!
interface management1
  ip address $ip_mgmt/24
```

```

!
interface loopback0
  ip address $ip_lo0/32
!
interface ethernet49/1
  speed forced 40gfull
  ip address $ip_to_spine1/31
!
interface ethernet50/1
  speed forced 40gfull
  ip address $ip_to_spine2/31
!
router bgp $bgp_as
  router-id $ip_lo0
  neighbor $ip_spine1 remote-as 65000
  neighbor $ip_spine2 remote-as 65000
  network $ip_lo0/32

```

We will write a Python script to read this file. Create a Python script file and call it as config_generator.py. with open("file_name") is used to open the file. It automatically closes the file after the indented code block is executed.

```

file_path = "/Users/anees/Dropbox/my-scripts/files/"
file_name = "config_template.txt"

config_file = file_path + file_name

with open(config_file) as read_file:
    config_template = read_file.read()

print config_template

```

Save and run the script.

```

anees:my-scripts anees$ python config_generator.py
hostname $hostname
!
interface management1
  ip address $ip_mgmt/24
!
interface loopback0
  ip address $ip_lo0/32
!
interface ethernet49/1
  speed forced 40gfull
  ip address $ip_to_spine1/31
!
interface ethernet50/1
  speed forced 40gfull
  ip address $ip_to_spine2/31
!
router bgp $bgp_as
  router-id $ip_lo0
  neighbor $ip_spine1 remote-as 65000
  neighbor $ip_spine2 remote-as 65000
  network $ip_lo0/32
!

```

Now, we are going to identify all the variables in the configuration template. We can use the regular expression module

to identify all the variables starting with \$. `re.findall("$w+", config_template)` creates a list of the words starting with \$. Since the configuration template may use the same variable name in multiple places, there will be duplicate entries in the list created by `re.findall`. We will convert the list to set to eliminate the duplicate entries.

```
import re

file_path = "/Users/anees/Dropbox/my-scripts/files/"
file_name = "config_template.txt"

config_file = file_path + file_name

with open(config_file) as read_file:
    config_template = read_file.read()

variables = set(re.findall("\$\w+", config_template))

print variables
```

Save and run the script.

```
anees:my-scripts anees$ python config_generator.py
set(['$ip_mgmt', '$ip_lo0', '$hostname', '$ip_spine2', '$ip_spine1', '$ip_to_spine2',
↪ '$ip_to_spine1', '$bgp_as'])
```

JSON

JSON provides a data structure which is easy to read by human as well as easy to parse the data using programming languages. The structure is very similar to what we learned in Python dictionary ({key: value}) earlier in this chapter. We will create a file using JSON format with the variables we have used in the configuration template as keys.

```
anees:my-scripts anees$ vi variables.json
{
  "$ip_mgmt": "192.168.1.11",
  "$ip_lo0": "10.10.10.10",
  "$hostname": "sjcpodl1or1",
  "$ip_spine2": "10.10.101.2",
  "$ip_spine1": "10.10.102.2",
  "$ip_to_spine2": "10.10.101.1",
  "$ip_to_spine1": "10.10.102.1",
  "$bgp_as": "65101"
}
```

We will update our `config_generator.py` to read this json file and replace the variables in the configuration template with the values in the `variables.json` file. In order to read the json file, we will use the Python json module.

```
import re
import json

file_path = "/Users/anees/Dropbox/my-scripts/files/"
file_name = "config_template.txt"
variables_file_name = "variables.json"

config_file = file_path + file_name
variables_file = file_path + variables_file_name

# Read Config template file
with open(config_file) as read_file:
```

```

    config_template = read_file.read()

# Read the variables from the config template
variables = set(re.findall("\$\w+", config_template))

# Read the variables from the json file
with open(variables_file) as read_file:
    variables_dictionary = json.load(read_file)

print variables_dictionary

```

Save and run the script.

```

anees:my-scripts anees$ python config_generator.py
{u'$ip_mgmt': u'192.168.1.11', u'$ip_lo0': u'10.10.10.10', u'$hostname': u'sjcpod1tor1
↪', u'$ip_spine2': u'10.10.101.2', u'$ip_spine1': u'10.10.102.2', u'$ip_to_spine2': u
↪'10.10.101.1', u'$ip_to_spine1': u'10.10.102.1', u'$bgp_as': u'65101'}

```

As you can see, `json.load(read_file)` imports the content of the json file as dictionary. Now we will update our script which generates the configuration from the configuration template and the variables dictionary.

```

import re
import json

file_path = "/Users/anees/Dropbox/my-scripts/files/"
file_name = "config_template.txt"
variables_file_name = "variables.json"

config_file = file_path + file_name
variables_file = file_path + variables_file_name

# Read Config template file
with open(config_file) as read_file:
    config_template = read_file.read()

# Read the variables from the config template
variables = set(re.findall("\$\w+", config_template))

# Read the variables from the json file
with open(variables_file) as read_file:
    variables_dictionary = json.load(read_file)

# Generate Configuration from Config template and variables
config = config_template

for each_variable in variables:
    config = config.replace(each_variable, variables_dictionary[each_variable])

print config

```

Save and run the script.

```

anees:my-scripts anees$ python config_generator.py
hostname sjcpod1tor1
!
interface management1
  ip address 192.168.1.11/24
!

```

```
interface loopback0
  ip address 10.10.10.10/32
!
interface ethernet49/1
  speed forced 40gfull
  ip address 10.10.102.1/31
!
interface ethernet50/1
  speed forced 40gfull
  ip address 10.10.101.1/31
!
router bgp 65101
  router-id 10.10.10.10
  neighbor 10.10.102.2 remote-as 65000
  neighbor 10.10.101.2 remote-as 65000
  network 10.10.10.10/32
!
```

YAML

YAML is a data serialization language which provides a human readable data structure that can be easily accessible by programming languages. YAML is a superset of JSON.

In the previous section, we created the variables file in JSON format. In this section, we will create the variables in YAML format. We are going to create the variables for multiple devices.

```
anees@my-scripts anees$ vi variables.yaml
---
JPE14080457:
  $ip_mgmt: 192.168.1.11
  $ip_lo0: 10.10.10.11
  $hostname: sjcpod1tor1
  $ip_spine2: 10.10.101.1
  $ip_spine1: 10.10.102.1
  $ip_to_spine2: 10.10.101.0
  $ip_to_spine1: 10.10.102.0
  $bgp_as: 65101

JPE14421537:
  $ip_mgmt: 192.168.1.12
  $ip_lo0: 10.10.10.12
  $hostname: sjcpod1tor2
  $ip_spine2: 10.10.101.3
  $ip_spine1: 10.10.102.3
  $ip_to_spine2: 10.10.101.2
  $ip_to_spine1: 10.10.102.2
  $bgp_as: 65102
```

We will update our `config_generator.py` script to read the yaml file and display the content. We will use `pprint` module to print the output.

```
import re
import yaml
import pprint

file_path = "/Users/anees/Dropbox/my-scripts/files/"
file_name = "config_template.txt"
```



```

variables_file_name = "variables.yaml"

config_file = file_path + file_name
variables_file = file_path + variables_file_name

# Read Config template file
with open(config_file) as read_file:
    config_template = read_file.read()

# Read the variables from the config template
variables = set(re.findall("\$\w+", config_template))

# Read the variables from the json file
with open(variables_file) as read_file:
    variables_dictionary = yaml.load(read_file)

pprint.pprint(variables_dictionary)

```

Save and run the script.

```

anees:my-scripts anees$ python config_generator.py
{'JPE14080457': {'$bgp_as': 65101,
                  '$hostname': 'sjcpod1tor1',
                  '$ip_lo0': '10.10.10.11',
                  '$ip_mgmt': '192.168.1.11',
                  '$ip_spine1': '10.10.102.1',
                  '$ip_spine2': '10.10.101.1',
                  '$ip_to_spine1': '10.10.102.0',
                  '$ip_to_spine2': '10.10.101.0'},
 'JPE14421537': {'$bgp_as': 65102,
                  '$hostname': 'sjcpod1tor2',
                  '$ip_lo0': '10.10.10.12',
                  '$ip_mgmt': '192.168.1.12',
                  '$ip_spine1': '10.10.102.3',
                  '$ip_spine2': '10.10.101.3',
                  '$ip_to_spine1': '10.10.102.2',
                  '$ip_to_spine2': '10.10.101.2'}}

```

Now we will update our script which generates the configuration from the configuration template and the variables dictionary.

```

import re
import yaml

file_path = "/Users/anees/Dropbox/my-scripts/files/"
file_name = "config_template.txt"
variables_file_name = "variables.yaml"

config_file = file_path + file_name
variables_file = file_path + variables_file_name

# Read Config template file
with open(config_file) as read_file:
    config_template = read_file.read()

# Read the variables from the config template
variables = set(re.findall("\$\w+", config_template))

```

```
# Read the variables from the json file
with open(variables_file) as read_file:
    variables_dictionary = yaml.load(read_file)

# Generate Configuration from Config template and variables
for each_switch in variables_dictionary:
    config = config_template
    for each_variable in variables:
        config = config.replace(each_variable, str(variables_dictionary[each_
↪switch][each_variable]))

    print "#" * 50
    print config
```

Save and run the script.

```
anees:my-scripts anees$ python config_generator.py
#####
hostname sjcpodl1tor2
!
interface management1
 ip address 192.168.1.12/24
!
interface loopback0
 ip address 10.10.10.12/32
!
interface ethernet49/1
 speed forced 40gfull
 ip address 10.10.102.2/31
!
interface ethernet50/1
 speed forced 40gfull
 ip address 10.10.101.2/31
!
router bgp 65102
 router-id 10.10.10.12
 neighbor 10.10.102.3 remote-as 65000
 neighbor 10.10.101.3 remote-as 65000
 network 10.10.10.12/32
!
#####
hostname sjcpodl1tor1
!
interface management1
 ip address 192.168.1.11/24
!
interface loopback0
 ip address 10.10.10.11/32
!
interface ethernet49/1
 speed forced 40gfull
 ip address 10.10.102.0/31
!
interface ethernet50/1
 speed forced 40gfull
 ip address 10.10.101.0/31
!
router bgp 65101
```

```
router-id 10.10.10.11
neighbor 10.10.102.1 remote-as 65000
neighbor 10.10.101.1 remote-as 65000
network 10.10.10.11/32
!
anees:my-scripts anees$
```

Summary

Since we learned the core Python concepts, we will move forward to build scripts for Arista networking use cases. In the next chapter, we will build a framework which you can use to build Python scripts for networking use cases. We are going to use Arista's Pyeapi module to interact with Arista switches. We have also used jsonrpc for some of the use cases. Before proceeding to next chapter, install pyeapi module using pip on your system.

Chapter 2: Script Framework for Use Cases

- *First Script - Show version*
- *Second Script - Running Show Version on Multiple Switches*
 - *Using IDLE*
 - *for loop*
- *Third Script - Using External Input*
- *Fourth Script – Storing Result in a Dictionary*
- *Fifth Script – Handling Exceptions*
- *Summary – Script Framework*

This chapter is going to help you to build a Python script framework using some of the core concepts learned in the previous chapter. By using this framework you build some of the common network operational use cases such as inventory, capacity planning and troubleshooting network issues later in this book.

First Script - Show version

In this section, we will write a Python script using pyeapi and connect to one of the Arista switches and collect “show version” from the switch.

Make sure you enable management api on the Arista switch. Below is the sample configuration to enable management api when the management interface is configured under default vrf.

```
configure
!
interface Management1
    ip address 172.28.132.42/20
!
```

```
management api http-commands
no shutdown
!
```

Below is the sample configuration to enable management api when the management interface is configured under non default vrf.

```
configure
!
interface Management1
    vrf forwarding mgmt
    ip address 172.28.132.45/20
!
management api http-commands
no shutdown
vrf mgmt
no shutdown
```

Launch the Python interpreter from your system and connect to Arista switch using pyeapi. If you have not installed pyeapi module, refer *Installing Python Modules*.

```
anees:~ anees$ python
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> import pyeapi

>>> node = pyeapi.connect(transport="https", host="172.28.132.41", username="admin",
↳ password="admin", port=None)

>>> version = node.execute(["show version"])

>>> version
{'u'jsonrpc': u'2.0', u'result': [{u'memTotal': 8069504, u'version': u'4.15.2F', u
↳ 'internalVersion': u'4.15.2F-2663444.4152F', u'serialNumber': u'JPE14080457', u
↳ 'systemMacAddress': u'00:1c:73:57:4f:49', u'bootupTimestamp': 1446677012.2, u
↳ 'memFree': 4381616, u'modelName': u'DCS-7050SX-128-F', u'architecture': u'i386', u
↳ 'internalBuildId': u'b664b979-69d7-4157-9543-20278086874a', u'hardwareRevision': u
↳ '02.00'}]}, u'id': u'4522839056'}
```

Response of the output is stored as dictionary in the variable “version”. Dictionary is a Python data structure represented in { } bracket and the data is represented as key:<value> pair.

```
>>> type(version)
<type 'dict'>

>>> version.keys()
[u'jsonrpc', u'result', u'id']

>>> version["result"]
[{u'memTotal': 8069504, u'version': u'4.15.2F', u'internalVersion': u'4.15.2F-2663444.
↳ 4152F', u'serialNumber': u'JPE14080457', u'systemMacAddress': u'00:1c:73:57:4f:49',
↳ u'bootupTimestamp': 1446677012.2, u'memFree': 4381616, u'modelName': u'DCS-7050SX-
↳ 128-F', u'architecture': u'i386', u'internalBuildId': u'b664b979-69d7-4157-9543-
↳ 20278086874a', u'hardwareRevision': u'02.00'}]
```

Value of the key “result” is a list which is another Python data structure represents in square [] brackets.

```
>>> type(version["result"])
<type 'list'>
>>>
>>> len(version["result"])
1
>>> version["result"][0]
{'u'memTotal': 8069504, u'version': u'4.15.2F', u'internalVersion': u'4.15.2F-2663444.4152F', u'serialNumber': u'JPE14080457', u'systemMacAddress': u'00:1c:73:57:4f:49', u'bootupTimestamp': 1446677012.2, u'memFree': 4381616, u'modelName': u'DCS-7050SX-128-F', u'architecture': u'i386', u'internalBuildId': u'b664b979-69d7-4157-9543-20278086874a', u'hardwareRevision': u'02.00'}
```

Finally the desired data is accessible in the output of `version["result"][0]`. As you see the curly bracket, it is a Python dictionary data structure. You can access the desired data using the keys “version”, “modelName” or “serialNumber”.

```
>>> version["result"][0]["version"]
u'4.15.2F'

>>> # Data is represented in unicode format u' '. You can convert to string using
↳str().

>>> str(version["result"][0]["version"])
'4.15.2F'

>>> str(version["result"][0]["serialNumber"])
'JPE14080457'

>>> str(version["result"][0]["modelName"])
'DCS-7050SX-128-F'
```

As you have seen, the actual output of `show version` is stored as dictionary under List which is a value within a parent dictionary. If it is confusing, Python’s `pprint` module provides a good view of the nested data structure.

```
>>> import pprint

>>> pprint.pprint(version)
{'u'id': u'4522839056',
 u'jsonrpc': u'2.0',
 u'result': [{u'architecture': u'i386',
              u'bootupTimestamp': 1446677012.2,
              u'hardwareRevision': u'02.00',
              u'internalBuildId': u'b664b979-69d7-4157-9543-20278086874a',
              u'internalVersion': u'4.15.2F-2663444.4152F',
              u'memFree': 4381616,
              u'memTotal': 8069504,
              u'modelName': u'DCS-7050SX-128-F',
              u'serialNumber': u'JPE14080457',
              u'systemMacAddress': u'00:1c:73:57:4f:49',
              u'version': u'4.15.2F'}]}}

>>> version["result"][0]["serialNumber"]
u'JPE14080457'
```

Second Script - Running Show Version on Multiple Switches

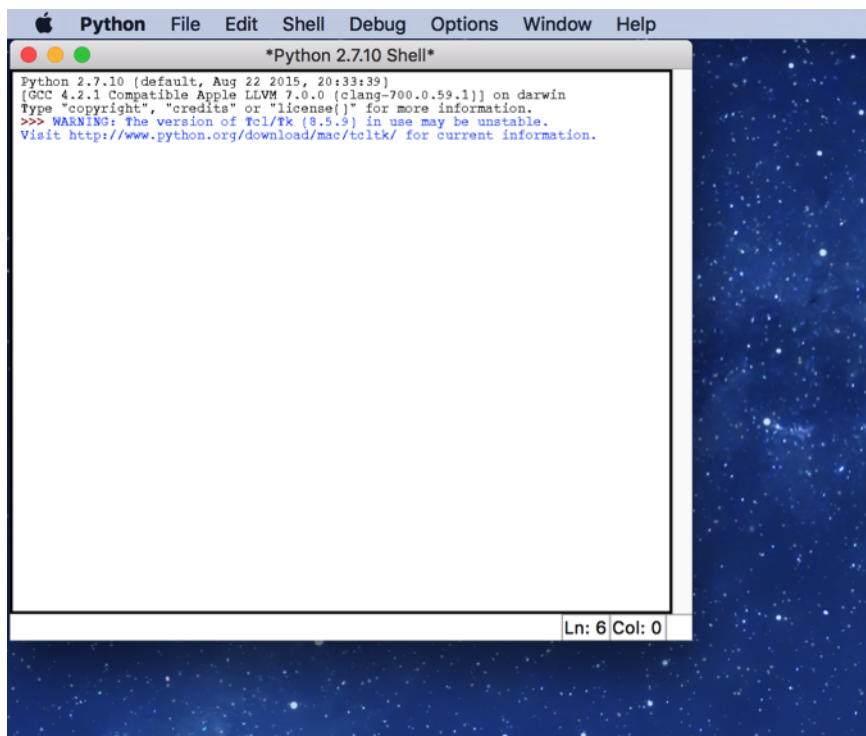
Power of scripting language is repeatability. Since you have already created a script to collect show version, let us use this code to collect show version from multiple switches in the network.

Since we will be creating several python scripts, let us create a folder in our systems. In this example I have created a folder called my-scripts under /Users/anees/Google Drive/.

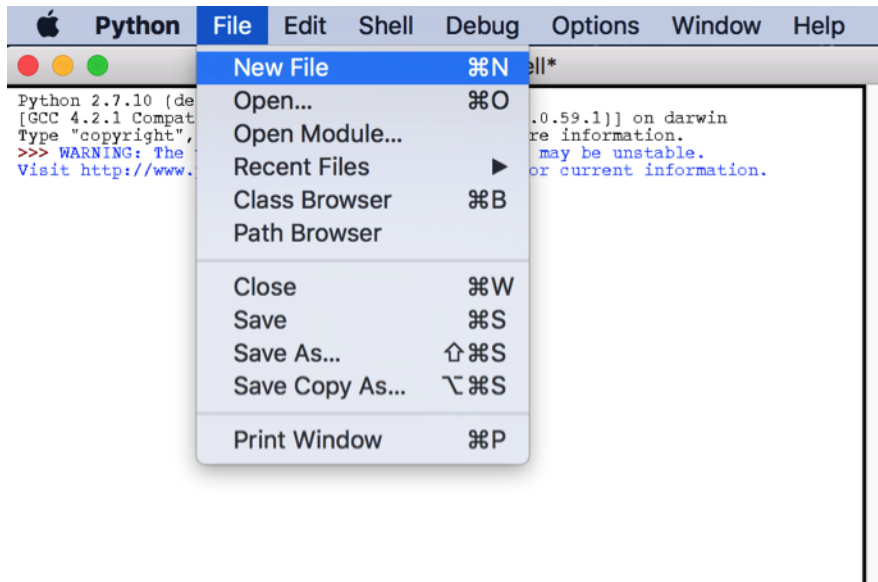
Using IDLE

Now it is the time to start using Python's IDLE environment. This is helpful while developing the script where we have to test the script as we make progress with the script.

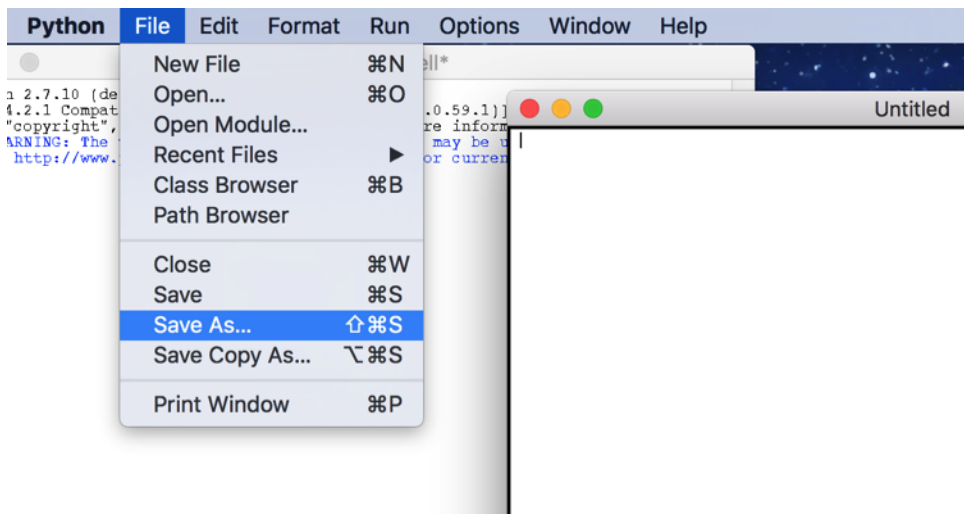
For MAC, type "idle" in the terminal window and you will see Python IDLE shell is opened.



Create a new file from File → New File



It opens a new untitled IDLE file. Save this file using File → Save As under the folder you created with the name as `inventory_version.py`.



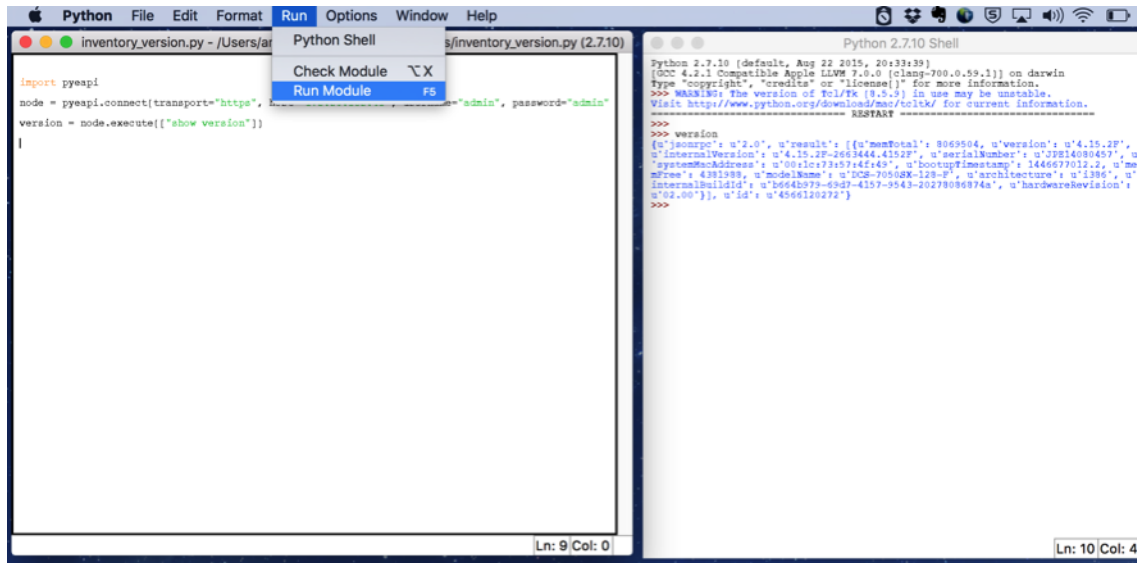
Write your script and save from File → Save (or Command + S).

```
import pyeapi

node = pyeapi.connect(transport="https", host="172.28.132.41", username="admin",
    password="admin", port=None)

version = node.execute(["show version"])
```

Test your script from Run → Run Module (or F5).



The script runs in the IDLE shell, which you can see, in the right side of the following screenshot. You can also access the variables from the IDLE shell.

for loop

Now let us create a script to collect the Model Name, Serial Number and EOS versions on multiple Arista switches in your network.

```
import pyeapi

# Create a List of Switches
switches = ["172.28.132.41", "172.28.132.40", "172.28.132.45"]

for switch in switches:
    # Define API Connection String
    node = pyeapi.connect(transport="https", host=switch, username="admin", password=
    ↪ "admin", port=None)

    # Execute the desired command
    version = node.execute(["show version"])

    # Print the desired Output
    print
    print ("*****")
    print ("Switch IP: %s" % (switch))
    print ("Model Name: %s" % (version["result"][0]["modelName"]))
    print ("Serial Number: %s" % (version["result"][0]["serialNumber"]))
    print ("EOS Version: %s" % (version["result"][0]["version"]))
    print ("*****")
    print
```

Save and run the command. You will see the output in the IDLE shell similar to the following output.

```
>>> ===== RESTART =====
>>>

*****
```

```

Switch IP: 172.28.132.41
Model Name: DCS-7050SX-128-F
Serial Number: JPE14080457
EOS Version: 4.15.2F
*****

*****

Switch IP: 172.28.132.40
Model Name: DCS-7050SX-128-F
Serial Number: JPE14080459
EOS Version: 4.15.2F
*****

*****

Switch IP: 172.28.132.45
Model Name: DCS-7280SE-64-F
Serial Number: JPE14443170
EOS Version: 4.15.2F
*****

```

Third Script - Using External Input

In the second script, we have listed the switch IP addresses, username and password in the script. In this script, we will enter the IPs in a text file and enforce the script to prompt for username and password when the script is executed.

Create a file named “switches” in the same directory as you are saving the Python scripts using any text editor of your choice. And add the list of IP addresses of the switches. Format the file as plain text (Format → Make Plain Text).

Open the IDLE and create a new python script named `inventory_version_input.py`.

```

'''

Third script - Inventory - show version - Using External Input

'''

import pyeapi

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

```

Run this command and verify the value of the variable `file` from the python shell.

```

Python 2.7.10 Shell

Python 2.7.10 (default, Aug 22 2015, 20:33:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.1)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
>>> file_switches
'/Users/anees/Google Drive/my-scripts/switches.txt'
>>>

```

Let us update the script to read the content of the file “switches”.

```
'''
Third script - Inventory - show version - Using External Input
'''

import pyeapi

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

with open(file_switches) as readfile:
    for line in readfile:
        print line
```

Run the script and verify the result.

```
>>> ===== RESTART =====
>>>
172.28.132.41
172.28.132.40
172.28.132.45
```

We will store the IP addresses read from the file into a list.

```
'''
Third script - Inventory - show version - Using External Input
'''

import pyeapi

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []

with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line)
```

Run this script.

```

>>> ===== RESTART =====
>>>

# You will not see any output since we are not sending any data to output

# Type the variable name switches

>>> switches
['172.28.132.41\n', '172.28.132.40\n', '172.28.132.45']

# It appears that new line character "\n" is added in the list

>>> type(switches)
<type 'list'>

>>> switches[0]
'172.28.132.41\n'

>>> switches[1]
'172.28.132.40\n'

# You can strip the new line character
>>> switches[0].strip()
'172.28.132.41'

```

Let us fix the script to strip the new line character.

```

'''
Third script - Inventory - show version - Using External Input
'''

import pyeapi

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []

with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

```

Save and run the script.

```

>>> ===== RESTART =====
>>>

>>> switches
['172.28.132.41', '172.28.132.40', '172.28.132.45']

```

Now let us get the username and password interactively instead of hard coding in the script.

```
'''
Third script - Inventory - show version - Using External Input
'''

import pyeapi

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []

with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

# Input Username and Password
my_username = raw_input("Enter your username: ")
my_password = raw_input("Enter your password: ")
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Enter your username: admin
Enter your password: admin
```

Obviously we don't want to display the password on the screen when we type. We will use the Python module "getpass" to input password without displaying on the screen.

```
'''
Third script - Inventory - show version - Using External Input
'''

import pyeapi
import getpass

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []

with open(file_switches) as readfile:
```

```

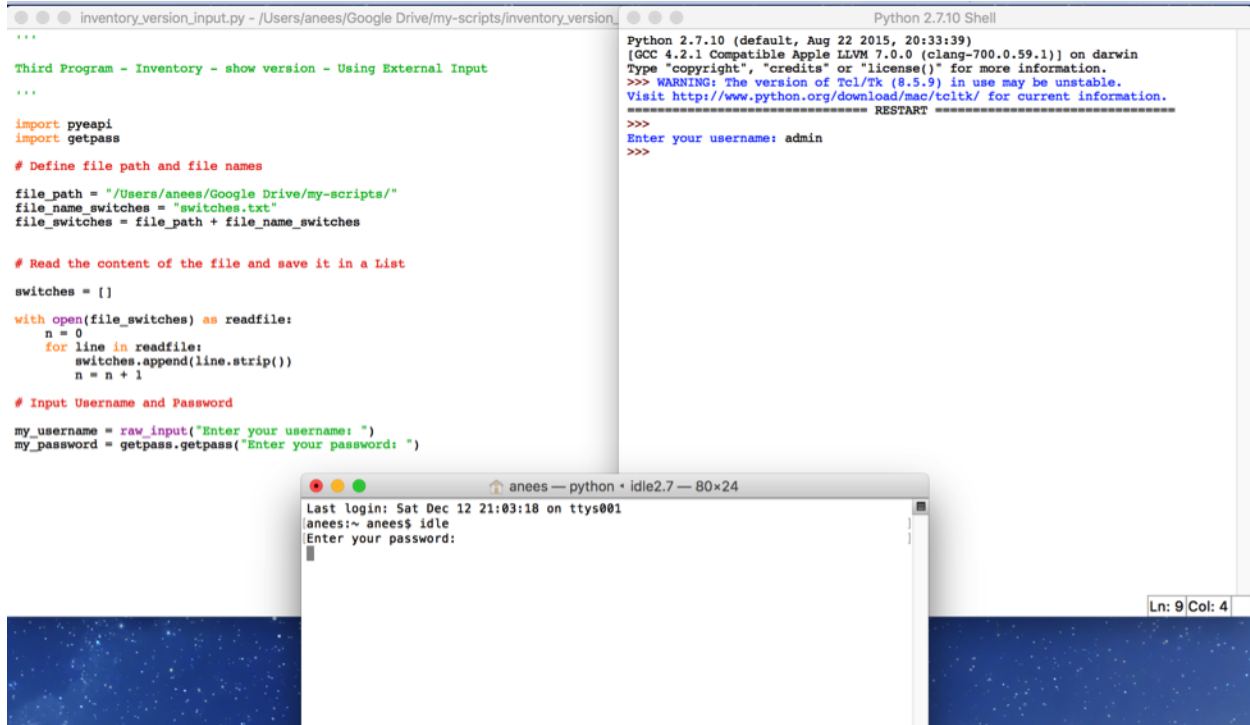
for line in readfile:
    switches.append(line.strip())

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

```

When you run this script from IDLE (on Apple Mac), the password prompt will not be prompted in the IDLE shell. It will be shown in the Apple Mac terminal where you have started the IDLE.



You can also run your Python script from terminal.

```

anees:my-scripts anees$ pwd
/Users/anees/Google Drive/my-scripts

anees:my-scripts anees$ ls
inventory_version.py      switches.txt
inventory_version_input.py

anees:my-scripts anees$ python inventory_version_input.py
Enter your username: admin
Enter your password:
anees:my-scripts anees$

```

Now we have achieved our requirements of inputting switch IP addresses, username and password from outside the python script. Let us complete the script with the inventory.

```

'''

Third script - Inventory - show version - Using External Input

```

```
'''

import pyeapi
import getpass

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []

with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

# Collect Inventory using pyeapi

for switch in switches:
    # Define API Connection String
    node = pyeapi.connect(transport="https", host=switch, username=my_username,
↳ password=my_password, port=None)

    # Execute the desired command
    version = node.execute(["show version"])

    # Print the desired Output
    print
    print ("*****")
    print ("Switch IP: %s" % (switch))
    print ("Model Name: %s" % (version["result"][0]["modelName"]))
    print ("Serial Number: %s" % (version["result"][0]["serialNumber"]))
    print ("EOS Version: %s" % (version["result"][0]["version"]))
    print ("*****")
    print
```

Run the script.

```
===== RESTART =====
>>>
Enter your username: admin

*****
Switch IP: 172.28.132.41
Model Name: DCS-7050SX-128-F
Serial Number: JPE14080457
EOS Version: 4.15.3F
*****
```



```

*****
Switch IP: 172.28.132.40
Model Name: DCS-7050SX-128-F
Serial Number: JPE14080459
EOS Version: 4.15.3F
*****

*****
Switch IP: 172.28.132.45
Model Name: DCS-7280SE-64-F
Serial Number: JPE14443170
EOS Version: 4.15.3F
*****

```

Fourth Script – Storing Result in a Dictionary

So far we output the data within the “for loop” as and when we parse the required data using the print statement. In this script we are going to save the output in a Python dictionary instead of printing within the “for loop”. At the end of the script we will print the dictionary using pprint.

Before writing the script, we need to come up with the structure of the dictionary. Structure of the dictionary depends on what data we want to collect and report. Our goal is to collect Model Name, Serial Number and EOS version of each of the switches. Hence the proposed dictionary structure is shown below:

```

{
  IP Address:
    {
      Model Name:
      Serial Number:
      EOS Version:
    }
}

```

Now, the algorithm is going to look like this

1. Create a blank dictionary before “for loop” `inventory = { }`
2. For each IP address, create an entry (Key, Value) with the IP address as the key and a blank dictionary as the value. `inventory["10.10.10.11"] = { }`
3. Add the entries for `inventory["10.10.10.11"]["Model Name:"]` `inventory["10.10.10.11"]["Serial Number:"]` `inventory["10.10.10.11"]["EOS Version:"]`

Launch Python interpreter from your terminal.

```

anees:~ anees$ python
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> inventory = { }
>>>
>>> type(inventory)
<type 'dict'>
>>>

```

```
>>> inventory["10.10.10.11"] = { }
>>>
>>> inventory
{'10.10.10.11': {}}
>>>
>>> inventory["10.10.10.11"]["Model Name:"] = "7050SX"
>>> inventory["10.10.10.11"]["Serial Number:"] = "srx123456"
>>> inventory["10.10.10.11"]["EOS Version:"] = "4.15.3f"
>>>
>>> inventory
{'10.10.10.11': {'Serial Number:': 'srx123456', 'EOS Version:': '4.15.3f', 'Model_
↪Name:': '7050SX'}}
>>>
>>> import pprint
>>>
>>> pprint.pprint(inventory)
{'10.10.10.11': {'EOS Version:': '4.15.3f',
                  'Model Name:': '7050SX',
                  'Serial Number:': 'srx123456'}}
```

Let us go back and update our original `inventory_version` script. Create a new python script with the name `inventory_version_output.py` and save it in your folder.

```
'''
Fourth script - Inventory - show version - Store Result in a Dictionary
'''

import pyeapi
import getpass
import pprint

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []

with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

# Collect Inventory using pyeapi

inventory = {}

for switch in switches:
```

```

inventory[switch] = {}

# Define API Connection String
node = pyeapi.connect(transport="https", host=switch, username=my_username,
↳password=my_password, port=None)

# Execute the desired command
version = node.execute(["show version"])

# Extract Desired data from show version
model_name = version["result"][0]["modelName"]
serial_number = version["result"][0]["serialNumber"]
EOS_version = version["result"][0]["version"]

# Update the inventory dictionary with the desired data
inventory[switch]["Model Name:"] = model_name
inventory[switch]["Serial Number:"] = serial_number
inventory[switch]["EOS Version:"] = EOS_version

pprint.pprint(inventory)

```

Run this script and verify the result.

```

>>> ===== RESTART =====
>>>
Enter your username: admin
{'172.28.132.40': {'EOS Version:': u'4.15.3F',
                  'Model Name:': u'DCS-7050SX-128-F',
                  'Serial Number:': u'JPE14080459'},
 '172.28.132.41': {'EOS Version:': u'4.15.3F',
                  'Model Name:': u'DCS-7050SX-128-F',
                  'Serial Number:': u'JPE14080457'},
 '172.28.132.45': {'EOS Version:': u'4.15.3F',
                  'Model Name:': u'DCS-7280SE-64-F',
                  'Serial Number:': u'JPE14443170'}}

```

Fifth Script – Handling Exceptions

On the switches.txt file, add an IP address of a switch that does not exist in the network or that does not have eAPI enabled. For example, the below list has the IP address 172.28.170.143 which does not exist in the network.

- 172.28.132.41
- **172.28.170.143**
- 172.28.132.40
- 172.28.132.45

Create a new script inventory_version_exception.py in your folder. Copy the script from inventory_version_output.py and run the script.

```

>>> ===== RESTART =====
>>>
Enter your username: admin
No handlers could be found for logger "pyeapi.eapilib"

```

```
Traceback (most recent call last):
  File "/Users/anees/Google Drive/my-scripts/inventory_version_exception.py", line 46,
  ↪ in <module>
    version = node.execute(["show version"])
  File "/Library/Python/2.7/site-packages/pyeapi/eapilib.py", line 464, in execute
    response = self.send(request)
  File "/Library/Python/2.7/site-packages/pyeapi/eapilib.py", line 394, in send
    raise ConnectionError(str(self), 'unable to connect to eAPI')
ConnectionError: unable to connect to eAPI
```

Because of that one incorrect IP address, the entire script fails. This will be annoying in real world where you may be managing hundreds of devices and any one device that is not available at the moment you are running the script make the entire script fail. Software scripting languages have a process called Exception Handling which should be used to react to any errors or exceptions that impacts the normal flow of your script.

Python has `try/except` keywords to handle exceptions so that you can run the scripts without exiting the script and reacts to the errors the way you wanted. The below example is used to explain `try/except` keywords.

```
try:
    inventory[switch] = {}
    # Define API Connection String
    node = pyeapi.connect(transport="https", host=switch, username=my_username,
    ↪ password=my_password, port=None)

    # Execute the desired command
    version = node.execute(["show version"])

    # Extract Desired data from show version
    model_name = version["result"][0]["modelName"]
    serial_number = version["result"][0]["serialNumber"]
    EOS_version = version["result"][0]["version"]

    # Update the inventory dictionary with the desired data

    inventory[switch]["Model Name:"] = model_name
    inventory[switch]["Serial Number:"] = serial_number
    inventory[switch]["EOS Version:"] = EOS_version

except:
    errors[switch] = "ConnectionError: unable to connect to eAPI"
```

The commands under the `try:` section called as try clause and the commands under `except:` section called as except clause. If there is any exception occurs while executing try clause, except clause will be executed and then the script execution continues. If there are no exceptions in the try clause, except clause is skipped.

We will update our script `inventory_version_exception.py` with `try/except` clause. In this script, we will create a separate dictionary called “errors” in which we will store the IP addresses of the switch that fails the `pyeapi` call and the corresponding error messages.

```
'''
Fifth script - Inventory - show version - Handling Exceptions
'''

import pyeapi
import getpass
import pprint
```

```

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []

with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

# Collect Inventory using pyeapi

# Collect Inventory using pyeapi

inventory = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
                               password=my_password, port=None)

        # Execute the desired command
        version = node.execute(["show version"])

        # Extract Desired data from show version
        model_name = version["result"][0]["modelName"]
        serial_number = version["result"][0]["serialNumber"]
        EOS_version = version["result"][0]["version"]

        # Update the inventory dictionary with the desired data
        inventory[switch] = {}
        inventory[switch]["Model Name:"] = model_name
        inventory[switch]["Serial Number:"] = serial_number
        inventory[switch]["EOS Version:"] = EOS_version

    except:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

pprint.pprint(errors)
pprint.pprint(inventory)

```

Save and run the script.

```

>>> ===== RESTART =====
>>>

```

```
Enter your username: admin
No handlers could be found for logger "pyeapi.eapilib"
{'172.28.170.143': 'ConnectionError: unable to connect to eAPI'}
{'172.28.132.40': {'EOS Version': 'u'4.15.3F',
                  'Model Name': 'u'DCS-7050SX-128-F',
                  'Serial Number': 'u'JPE14080459'},
 '172.28.132.41': {'EOS Version': 'u'4.15.3F',
                  'Model Name': 'u'DCS-7050SX-128-F',
                  'Serial Number': 'u'JPE14080457'},
 '172.28.132.45': {'EOS Version': 'u'4.15.3F',
                  'Model Name': 'u'DCS-7280SE-64-F',
                  'Serial Number': 'u'JPE14443170'}}
```

Exception can happen due to variety of reasons. For example, it could be because of the switch is not reachable or the switch is reachable but the EOS command entered in the script is incorrect. In that case, we are expecting the module (Pyeapi) that is used to facilitate the connection should differentiate the errors and allow the scriptmer to handle it in the script. Pyeapi module has few types of exceptions. For more information about the exceptions supported by pyeapi module, refer the pyeapi module documentation [Python Client for eAPI](#).

You can also explore the modules from Python interpreter.

```
>>> dir(pyeapi)
['__all__', '__author__', '__builtins__', '__doc__', '__file__', '__name__', '__
↳package__', '__path__', '__version__', 'client', 'config_for', 'connect', 'connect_
↳to', 'eapilib', 'load_config', 'utils']

>>> dir(pyeapi.eapilib)
['CommandError', 'ConnectionError', 'DEFAULT_HTTPS_PORT', 'DEFAULT_HTTP_LOCAL_PORT',
↳'DEFAULT_HTTP_PATH', 'DEFAULT_HTTP_PORT', 'DEFAULT_UNIX_SOCKET', 'EapiConnection',
↳'EapiError', 'HTTPConnection', 'HTTPSConnection', 'HttpConnection',
↳'HttpEapiConnection', 'HttpLocalEapiConnection', 'HttpsConnection',
↳'HttpsEapiConnection', 'SocketConnection', 'SocketEapiConnection', '_LOGGER', '__
↳builtins__', '__doc__', '__file__', '__name__', '__package__', 'base64', 'debug',
↳'https_connection_factory', 'json', 'logging', 'make_iterable', 'socket', 'ssl',
↳'sys']
```

We can update our script `inventory_version_exception.py` to differentiate the type of pyeapi exceptions.

```
'''
Fifth script - Inventory - show version - Handling Exceptions
'''

import pyeapi
import getpass
import pprint

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List
```

```

switches = []

with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

# Collect Inventory using pyeapi

inventory = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        ↪password=my_password, port=None)

        # Execute the desired command
        version = node.execute(["show version"])

        # Extract Desired data from show version
        model_name = version["result"][0]["modelName"]
        serial_number = version["result"][0]["serialNumber"]
        EOS_version = version["result"][0]["version"]

        # Update the inventory dictionary with the desired data
        inventory[switch] = {}
        inventory[switch]["Model Name:"] = model_name
        inventory[switch]["Serial Number:"] = serial_number
        inventory[switch]["EOS Version:"] = EOS_version

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

pprint.pprint(errors)
pprint.pprint(inventory)

```

Verify the error messages.

```

>>> ===== RESTART =====
>>>
Enter your username: admin
No handlers could be found for logger "pyeapi.eapilib"
{'172.28.170.143': 'ConnectionError: unable to connect to eAPI'}
{'172.28.132.40': {'EOS Version:': u'4.15.3F',
                  'Model Name:': u'DCS-7050SX-128-F',
                  'Serial Number:': u'JPE14080459'}},
{'172.28.132.41': {'EOS Version:': u'4.15.3F',
                  'Model Name:': u'DCS-7050SX-128-F',
                  'Serial Number:': u'JPE14080457'}},

```

```
'172.28.132.45': {'EOS Version:': u'4.15.3F',
                  'Model Name:': u'DCS-7280SE-64-F',
                  'Serial Number:': u'JPE14443170'}}
```

Change the command syntax “show version” to “show ver” and run the command.

```
>>> ===== RESTART =====
>>>
Enter your username: admin
No handlers could be found for logger "pyeapi.eapilib"
{'172.28.132.40': 'CommandError: Check your EOS command syntax',
 '172.28.132.41': 'CommandError: Check your EOS command syntax',
 '172.28.132.45': 'CommandError: Check your EOS command syntax',
 '172.28.170.143': 'ConnectionError: unable to connect to eAPI'}
```

Summary – Script Framework

We have learned various Python concepts step by step using the inventory use case and finally we got a structure for our network use case Python script if you have observed closely all the five scripts. The structure of code is shown below:

Section 1: Import Modules

```
import pyeapi
import getpass
import pprint
```

Section 2: Read the list of switch IP addresses from a file and store it in a Python list

```
file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

switches = []

with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())
```

Section 3: Input Username and Password that have access to the switches

```
my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")
```

Section 4: Main Algorithm Specific to your use case

For every switch in the Python list

1. Connect to the switch using Arista’s pyeapi module
2. Execute the commands needed for your logic
3. Core Logic
4. Store the result into a dictionary


```

inventory = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        ↪password=my_password, port=None)

        # Execute the desired command
        version = node.execute(["show version"])

        # Extract Desired data from show version
        model_name = version["result"][0]["modelName"]
        serial_number = version["result"][0]["serialNumber"]
        EOS_version = version["result"][0]["version"]

        # Update the inventory dictionary with the desired data
        inventory[switch] = {}
        inventory[switch]["Model Name:"] = model_name
        inventory[switch]["Serial Number:"] = serial_number
        inventory[switch]["EOS Version:"] = EOS_version

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

```

Section 5: Print the dictionaries

```

pprint.pprint(errors)
pprint.pprint(inventory)

```

We are going to use this five sections framework for all the use cases in this document. In all the use cases we reuse the commands from sections 1, 2, 3, 4A and 5. Only sections that changes based on the requirements of use cases are 4B, 4C, and 4D.

Chapter 3: Troubleshooting Use Cases

- *Monitor Data Plane Drops*
 - *Arista EOS Show Commands*
 - *Develop Script*
 - *Final Script*
- *Monitor Control Plane Drops*
 - *Arista EOS Show Commands*
 - *Develop Script*
 - *Final Script*

This chapter shows how to build Python scripts for a couple of network troubleshooting use cases. There are no new Python concepts introduced in this chapter but it gives a good practice to use the framework and Python concepts you learned so far in this book. For all the use cases, first we are going to write a high level troubleshooting steps, then we will start writing the script step by step based on the troubleshooting steps.

Monitor Data Plane Drops

The goal of this script is to discover if there are any packet drops in any of the switches in the network. First we will write down the troubleshooting steps and then we will build the script.

Arista EOS Show Commands

Our requirement is to list out the name of the switches, interface numbers and the corresponding drop counters. You can collect all the information using “show interfaces” command. In this example, we are going to use “show interfaces

counters discards” to find the interfaces that see drops and then collect the “show interface” for that particular interface and get the detailed drop counters.

Step 1: Identify if there are any drops in the switch and obtain the interface numbers.

```
Arista-switch# show interfaces counters discards | json
{
  "inDiscardsTotal": 0,
  "interfaces": {
    "Ethernet8": {
      "outDiscards": 0,
      "inDiscards": 0
    },
    "Ethernet9": {
      "outDiscards": 0,
      "inDiscards": 0
    },
    "Ethernet2": {
      "outDiscards": 0,
      "inDiscards": 0
    },
    "Ethernet3": {
      "outDiscards": 0,
      "inDiscards": 0
    },
    "Ethernet1": {
      "outDiscards": 0,
      "inDiscards": 0
    },
    "Ethernet21": {
      "outDiscards": 45941,
      "inDiscards": 0
    }
  },
}
```

Step 2: If any of the interfaces has non-zero counter in inDiscards or outDiscards, collect the show interface and print the detailed output or input error counters.

```
Arista-switch# show interfaces ethernet 21 | json
{
  "interfaces": {
    "Ethernet21": {
      "lastStatusChangeTimestamp": 1450734922.8865843,
      "name": "Ethernet21",
      "interfaceStatus": "connected",
      "autoNegotiate": "success",
      "burnedInAddress": "00:1c:73:57:4f:5e",
      "loopbackMode": "loopbackNone",
      "interfaceStatistics": {
        "inBitsRate": 0.7396139208713536,
        "inPktsRate": 0.0007222792196009312,
        "outBitsRate": 549.4475135300596,
        "updateInterval": 5.0,
        "outPktsRate": 0.9075684364502475
      },
      "mtu": 9214,
      "hardware": "ethernet",
      "duplex": "duplexFull",
      "bandwidth": 1000000000,
      "forwardingModel": "dataLink",
    }
  },
}
```

```

        "lineProtocolStatus": "up",
        "interfaceCounters": {
            "outBroadcastPkts": 11025,
            "linkStatusChanges": 5,
            "totalOutErrors": 0,
            "inMulticastPkts": 754,
            "counterRefreshTime": 1450757484.079082,
            "inBroadcastPkts": 0,
            "outputErrorsDetail": {
                "deferredTransmissions": 0,
                "txPause": 0,
                "collisions": 0,
                "lateCollisions": 0
            },
            "inOctets": 96512,
            "outDiscards": 45941,
            "outOctets": 78324214888,
            "inUcastPkts": 0,
            "inputErrorsDetail": {
                "runtFrames": 0,
                "rxPause": 0,
                "fcsErrors": 0,
                "alignmentErrors": 0,
                "giantFrames": 0,
                "symbolErrors": 0
            },
            "outUcastPkts": 152941271,
            "outMulticastPkts": 1512,
            "totalInErrors": 0,
            "inDiscards": 0
        },
        "interfaceMembership": "Member of Port-Channel10",
        "interfaceAddress": [],
        "physicalAddress": "00:1c:73:57:4f:5e",
        "description": "F5-2"
    }
}

```

Develop Script

Open the IDLE and create a new script and save as `interface_drops.py` in your folder. Copy the code from section 1, 2, 3 from our framework and paste it in this new script.

```

"""
Discover if there is any packet drops in the network
"""

### Section 1

import pyeapi
import getpass
import pprint

### Section 2

```

```
# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []
with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

### Section 3

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")
```

Step 1: Identify interfaces having packet drops

Start section 4 with the usual `for` loop and `pyeapi` command. Collect “show interfaces counters discards” output from the switches. Later in this script we will store the result in a dictionary we will name as `interface_drops`.

```
### Section 4

interface_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        ↪password=my_password, port=None)

        # Execute the desired command
        interface_counters = node.execute(["show interfaces counters discards"])

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

pprint.pprint(errors)
```

Save and run the script. We are going to explore the dictionary to find the exact key to see the the packet drops.

```
===== RESTART =====
>>>
Enter your username: admin
>>>
>>> pprint.pprint(interface_counters)
{'u'id': u'4408635024',
```

```

u'jsonrpc': u'2.0',
u'result': [{u'inDiscardsTotal': 0,
            u'interfaces': {u'Ethernet1/1': {u'inDiscards': 0,
                                              u'outDiscards': 0},
                           u'Ethernet10/1': {u'inDiscards': 0,
                                              u'outDiscards': 0},
                           u'Ethernet10/2': {u'inDiscards': 0,
                                              u'outDiscards': 0},
                           u'Ethernet10/3': {u'inDiscards': 0,
                                              u'outDiscards': 0}}}]

>>> pprint.pprint(interface_counters["result"])
[{u'inDiscardsTotal': 0,
  u'interfaces': {u'Ethernet1/1': {u'inDiscards': 0, u'outDiscards': 0},
                  u'Ethernet10/1': {u'inDiscards': 0, u'outDiscards': 0},
                  u'Ethernet10/2': {u'inDiscards': 0, u'outDiscards': 0},
                  u'Ethernet10/3': {u'inDiscards': 0, u'outDiscards': 0},
                  u'Ethernet10/4': {u'inDiscards': 0, u'outDiscards': 0},
                  u'Ethernet11/1': {u'inDiscards': 0, u'outDiscards': 0}}}]

>>> pprint.pprint(interface_counters["result"][0])
{u'inDiscardsTotal': 0,
 u'interfaces': {u'Ethernet1/1': {u'inDiscards': 0, u'outDiscards': 0},
                 u'Ethernet10/1': {u'inDiscards': 0, u'outDiscards': 0},
                 u'Ethernet10/2': {u'inDiscards': 0, u'outDiscards': 0},
                 u'Ethernet10/3': {u'inDiscards': 0, u'outDiscards': 0},
                 u'Ethernet10/4': {u'inDiscards': 0, u'outDiscards': 0},
                 u'Ethernet11/1': {u'inDiscards': 0, u'outDiscards': 0}}}

>>> pprint.pprint(interface_counters["result"][0]["interfaces"])
{u'Ethernet1/1': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet10/1': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet10/2': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet10/3': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet10/4': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet11/1': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet11/2': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet11/3': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet11/4': {u'inDiscards': 0, u'outDiscards': 0},
 u'Ethernet12/1': {u'inDiscards': 0, u'outDiscards': 0}}

>>> interface_counters_clean = interface_counters["result"][0]["interfaces"]
>>>
>>> pprint.pprint(interface_counters_clean.keys())
[u'Ethernet4/3',
 u'Ethernet55/4',
 u'Ethernet15/4',
 u'Ethernet26/3',
 u'Ethernet15/2',
 u'Ethernet54/1',
 u'Ethernet48/4',
 u'Ethernet58/4',
 u'Ethernet48/1',
 u'Ethernet48/2',
 u'Ethernet48/3']

```

```
u'Ethernet58/3',
u'Ethernet36/3',
u'Ethernet36/2',

>>> interface_counters_clean["Ethernet4/3"]
{u'inDiscards': 0, u'outDiscards': 0}
>>> interface_counters_clean["Ethernet4/3"]["inDiscards"]
0
>>> interface_counters_clean["Ethernet4/3"]["outDiscards"]
0
```

For every result from a pyeapi call, our interesting data is under interface_counters [“results”] [0] [“interfaces”] key. Now we know the counters to see the packet drops which is interface_counters_clean [<interface>] [“inDiscards”].

Step 2: Collect the Input/Output Drops Statistics

We are going to iterate through the interfaces within each switch and look for non zero counters. If we see a non zero counter, we will initiate a pyeapi call to collect the “show interface” for that specific interface.

```
### Section 4

interface_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        password=my_password, port=None)

        # Execute the desired command
        interface_counters = node.execute(["show interfaces counters discards"])
        interface_counters_clean = interface_counters["result"][0]["interfaces"]

        """
        Parse through each interface and see if there any non zero inDiscards or
        outDiscards.
        If any interface has non zero counters, collect "show interface" output of
        that interface
        """
        for interface in interface_counters_clean.keys():
            if interface_counters_clean[interface]["inDiscards"] or interface_
            counters_clean[interface]["outDiscards"] != 0:
                show_interface = node.execute(["show interfaces " + str(interface)])

        except pyeapi.eapilib.ConnectionError:
            errors[switch] = "ConnectionError: unable to connect to eAPI"

        except pyeapi.eapilib.CommandError:
            errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

pprint.pprint(errors)
```

Save and run the script. Now we are going to explore within the “show interface” output, what are the specific counters to collect and report.


```
>>> pprint.pprint(show_interface)
{'id': u'4471515088',
 'jsonrpc': u'2.0',
 'result': [{u'interfaces': {u'Ethernet21': {u'autoNegotiate': u'success',
                                             u'bandwidth': 1000000000,
                                             u'burnedInAddress': u'00:1c:73:57:4f:5e',
                                             u'description': u'F5-2',
                                             u'duplex': u'duplexFull',
                                             u'forwardingModel': u'dataLink',
                                             u'hardware': u'ethernet',
                                             u'interfaceAddress': [],
                                             u'interfaceCounters': {u'counterRefreshTime': 1450763597.259004,
                                                                     u'inBroadcastPkts': 0,
                                                                     u'inDiscards': 0,
                                                                     u'inMulticastPkts': 958,
                                                                     u'inOctets': 122624,
                                                                     u'inUcastPkts': 0,
                                                                     u'inputErrorsDetail': {u'alignmentErrors': 0,
                                                                     u'fcsErrors': 0,
                                                                     u'giantFrames': 0,
                                                                     u'runtFrames': 0,
                                                                     u'rxPause': 0,
                                                                     u'symbolErrors': 0},
                                                                     u'linkStatusChanges': 5,
                                                                     u'outBroadcastPkts': 14502,
                                                                     u'outDiscards': 45941,
                                                                     u'outMulticastPkts': 1919,
                                                                     u'outOctets': 78324506444,
                                                                     u'outUcastPkts': 152941271,
                                                                     u'outputErrorsDetail': {u'collisions': 0,
                                                                     u'deferredTransmissions': 0,
                                                                     u'lateCollisions': 0,
                                                                     u'txPause': 0},
                                                                     u'totalInErrors': 0,
                                                                     u'totalOutErrors': 0},
                                                                     u'interfaceMembership': u'Member of Port-Channel10',
                                                                     u'interfaceStatistics': {u'inBitsRate': 5.9562810739578245,
                                                                     u'inPktsRate': 0.005816680736286938,
                                                                     u'outBitsRate': 723.3999156647525,
                                                                     u'outPktsRate': 1.4015494404625337,
                                                                     u'updateInterval': 5.0},
                                                                     u'interfaceStatus': u'connected',
                                                                     u'lastStatusChangeTimestamp': 1450734922.886585,
                                                                     u'lineProtocolStatus': u'up',
                                                                     u'loopbackMode': u'loopbackNone',
                                                                     u'mtu': 9214,
                                                                     u'name': u'Ethernet21',
                                                                     u'physicalAddress': u'00:1c:73:57:4f:5e'}}}}}]
|
>>>
>>>
```

```
>>> show_interface["result"][0]["interfaces"]["Ethernet21"]["interfaceCounters"][
↪ "inputErrorsDetail"]

{'runtFrames': 0, u'fcsErrors': 0, u'alignmentErrors': 0, u'rxPause': 0, u
↪ 'symbolErrors': 0, u'giantFrames': 0}

>>> show_interface["result"][0]["interfaces"]["Ethernet21"]["interfaceCounters"][
↪ "outputErrorsDetail"]

{'lateCollisions': 0, u'deferredTransmissions': 0, u'txPause': 0, u'collisions': 0}
```

Now we know the key for the input and output error details. What we need to know is to whether to collect the input or output error details. You can use if statements to collect input or output error details depends on whether you see input or output discards.

```
### Section 4

interface_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
↪ password=my_password, port=None)
```

```
# Execute the desired command
interface_counters = node.execute(["show interfaces counters discards"])
interface_counters_clean = interface_counters["result"][0]["interfaces"]

"""
Parse through each interface and see if there any non zero inDiscards or
↳outDiscards.
If any interface has non zero counters, collect "show interface" ouput of
↳that interface
"""
for interface in interface_counters_clean.keys():
    if interface_counters_clean[interface]["inDiscards"] or interface_
↳counters_clean[interface]["outDiscards"] != 0:
        show_interface = node.execute(["show interfaces " + str(interface)])
        show_interface_clean = show_interface["result"][0]["interfaces
↳"][interface]["interfaceCounters"]
        if interface_counters_clean[interface]["inDiscards"] != 0:
            print show_interface_clean["inputErrorsDetail"]
        if interface_counters_clean[interface]["outDiscards"] != 0:
            print show_interface_clean["outputErrorsDetail"]

except pyeapi.eapilib.ConnectionError:
    errors[switch] = "ConnectionError: unable to connect to eAPI"

except pyeapi.eapilib.CommandError:
    errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

pprint.pprint(errors)
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Enter your username: admin

Traceback (most recent call last):
  File "/Users/anees/Google Drive/my-scripts/interface_drops.py", line 60, in <module>
    print show_interface_clean["outputErrorsDetail"]
KeyError: 'outputErrorsDetail'
```

We are seeing an error message in the above output. It says that there is no key `outputErrorsDetail` in the `show interface <specific interface>`. If you `pprint` the `show interface <specific interface>` output, you can see that there are no `outputErrorsDetail` key under `interfaceCounters` section. This is because the output shown is for port channel interface. This specific counter is applicable only for physical interface. So we are going to add a check so that if the interface is port channel, we are not going to look for any counters.

```
>>> pprint.pprint(show_interface)
{'id': u'4475357200',
 'jsonrpc': u'2.0',
 'result': [{u'interfaces': {u'Port-Channel10': {u'bandwidth': 2000000000,
        u'description': u'',
        u'fallbackEnabled': False,
        u'fallbackEnabledType': u'fallbackNone',
        u'forwardingModel': u'bridged',
        u'hardware': u'portChannel',
        u'interfaceAddress': [],
        u'interfaceCounters': {u'counterRefreshTime': 1450766002.519597,
            u'inBroadcastPkts': 30570,
            u'inDiscards': 0,
            u'inMulticastPkts': 2092,
            u'inOctets': 2345856,
            u'inUcastPkts': 0,
            u'linkStatusChanges': 2,
            u'outBroadcastPkts': 25159,
            u'outDiscards': 45941,
            u'outMulticastPkts': 19506,
            u'outOctets': 156572394254,
            u'outUcastPkts': 305764360,
            u'totalInErrors': 0,
            u'totalOutErrors': 0},
        u'interfaceStatistics': {u'inBitsRate': 491.32858209660844,
            u'inPktsRate': 0.9002464696141602,
            u'outBitsRate': 593.3438837255551,
            u'outPktsRate': 0.7360843213048404,
            u'updateInterval': 5.0},
        u'interfaceStatus': u'connected',
        u'lastStatusChangeTimestamp': 1450734924.6027756,
        u'lineProtocolStatus': u'up',
        u'memberInterfaces': {u'Ethernet20': {u'bandwidth': 1000000000,
            u'duplex': u'duplexFull'},
            u'Ethernet21': {u'bandwidth': 1000000000,
            u'duplex': u'duplexFull'}},
        u'mtu': 9214,
        u'name': u'Port-Channel10',
        u'physicalAddress': u'00:1c:73:57:4f:5d'}}}]}}
```

We are going to do the interface check using “if ‘Port-Channel’ not in interface:” logic. It would be better idea to use this logic “if ‘Ethernet’ in interface:” instead of “‘Port-Channel’ not in:”. Because show interfaces will also contains SVIs. We need to look at only the Ethernet interfaces. However for learning perspective, we are going to use “if ‘Port-Channel’ not in interface:”.

```
### Section 4

interface_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        ↪password=my_password, port=None)

        # Execute the desired command
        interface_counters = node.execute(["show interfaces counters discards"])
        interface_counters_clean = interface_counters["result"][0]["interfaces"]

        """
        Parse through each interface and see if there any non zero inDiscards or
        ↪outDiscards.
        If any interface has non zero counters, collect "show interface" output of
        ↪that interface
        """
        for interface in interface_counters_clean.keys():
            if "Port-Channel" not in interface:
                if interface_counters_clean[interface]["inDiscards"] or interface_
                ↪counters_clean[interface]["outDiscards"] != 0:
                    show_interface = node.execute(["show interfaces " +
                    ↪str(interface)])
```

```
        show_interface_clean = show_interface["result"][0]["interfaces
↪"][interface]["interfaceCounters"]
        if interface_counters_clean[interface]["inDiscards"] != 0:
            print show_interface_clean["inputErrorsDetail"]
        if interface_counters_clean[interface]["outDiscards"] != 0:
            print show_interface_clean["outputErrorsDetail"]

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

pprint.pprint(errors)
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Enter your username: admin

{u'lateCollisions': 0, u'deferredTransmissions': 0, u'txPause': 0, u'collisions': 0}
{}
```

Step 3: Saving the Result in a Dictionary

Let us store the result in a nice format in a dictionary.

```
interface_drops =
{
Switch_host_name: {
    Interface_number: {
        "Interface status": <value>
        "Line Protocol Status": <value>
        "inDiscards": {
            "Total Discards": <value>
            InputErrorsDetail: {
                }
        }
        "outDiscards": {
            "Total Discards":
            "outputErrorsDetail":{
                }
        }
    }
}
```

First you need to initiate the dictionary when you start the section 4 in the script.

```
interface_drops = {}
for switch in switches:
```

For every switch, create a key, value pair in the interface_drops dictionary. Here, the value is another dictionary. Since it has to be created for each switch, we create this line inside the “for loop” of the switches. You need to get the hostname of the switch using the command “show hostname”.

```
interface_drops = {}
for switch in switches:
```

```

try:
    # lines skipped
    host_name = node.execute(["show hostname"])
    host_name_clean = str(host_name["result"][0]["hostname"])
    interface_drops[host_name_clean] = {}

```

For each switch, we need to create a key, value pair for the interface we are seeing packet drops. This statement should be inside the “for loop” of the interfaces of each switch. Since we need to create the key, value pair only if you see any packet drops, this line will be inside the if statement.

```

interface_drops = {}
for switch in switches:
    try:
        # lines skipped
        interface_drops[host_name_clean] = {}
        for interface in interface_counters_clean.keys():
            if interface_counters_clean[interface]["inDiscards"] or interface_
↪counters_clean[interface]["outDiscards"] != 0:
                interface_drops[host_name_clean][interface] = {}

```

If there is any input discards, we will record the total input discards and input error statistics.

```

if interface_counters_clean[interface]["inDiscards"] != 0:
    interface_drops[host_name_clean][interface]["inDiscards"] = {}
    interface_drops[host_name_clean][interface]["inDiscards"]["Total Discards"] = _
↪interface_counters_clean[interface]["inDiscards"]
    interface_drops[host_name_clean][interface]["inDiscards"]["Input Errors"] = show_
↪interface_clean["inputErrorsDetail"]

```

Similarly if there is any output discards, then we will record the total output discards and output error statistics.

```

if interface_counters_clean[interface]["outDiscards"] != 0:
    interface_drops[host_name_clean][interface]["outDiscards"] = {}
    interface_drops[host_name_clean][interface]["outDiscards"]["Total Discards"] = _
↪interface_counters_clean[interface]["outDiscards"]
    interface_drops[host_name_clean][interface]["outDiscards"]["Output Errors"] = _
↪show_interface_clean["outputErrorsDetail"]

```

Now let us look at the section 4 of the script we have developed so far.

```

### Section 4

interface_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username, _
↪password=my_password, port=None)

        # Execute the desired command
        interface_counters = node.execute(["show interfaces counters discards"])
        interface_counters_clean = interface_counters["result"][0]["interfaces"]

        # Collect hostname for documenting results under the host name
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])

```

```

interface_drops[host_name_clean] = {}

"""
    Parse through each interface and see if there any non zero inDiscards or
    outDiscards.
    If any interface has non zero counters, collect "show interface" output of
    that interface
"""

for interface in interface_counters_clean.keys():
    # Skip if the interface is port channel
    if "Port-Channel" not in interface:
        if interface_counters_clean[interface]["inDiscards"] or interface_
        counters_clean[interface]["outDiscards"] != 0:
            # Create an entry for the interface under the host name
            interface_drops[host_name_clean][interface] = {}

            # Collect the interface statistics from the switch
            show_interface = node.execute(["show interfaces " +
            str(interface)])
            show_interface_clean = show_interface["result"][0]["interfaces
            "][interface]["interfaceCounters"]

            # Collect interface and line protocol status just for
            documentation purpose
            interface_drops[host_name_clean][interface]["Interface Status"] =
            show_interface["result"][0]["interfaces"][interface]["interfaceStatus"]
            interface_drops[host_name_clean][interface]["Line Protocol Status
            "] = show_interface["result"][0]["interfaces"][interface]["lineProtocolStatus"]

            # Collect detailed input or output drop counters if there are
            input or output drops
            if interface_counters_clean[interface]["inDiscards"] != 0:
                interface_drops[host_name_clean][interface]["inDiscards"] = {}
                interface_drops[host_name_clean][interface]["inDiscards"][
                "Total Discards"] = interface_counters_clean[interface]["inDiscards"]
                interface_drops[host_name_clean][interface]["inDiscards"][
                "Input Errors"] = show_interface_clean["inputErrorsDetail"]
            if interface_counters_clean[interface]["outDiscards"] != 0:
                interface_drops[host_name_clean][interface]["outDiscards"] =
                {}
                interface_drops[host_name_clean][interface]["outDiscards"][
                "Total Discards"] = interface_counters_clean[interface]["outDiscards"]
                interface_drops[host_name_clean][interface]["outDiscards"][
                "Output Errors"] = show_interface_clean["outputErrorsDetail"]

        except pyeapi.eapilib.ConnectionError:
            errors[switch] = "ConnectionError: unable to connect to eAPI"

        except pyeapi.eapilib.CommandError:
            errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

pprint.pprint(errors)
pprint.pprint(interface_drops)

```

Save and run the script.

```

>>> ===== RESTART =====
>>>
Enter your username: admin
{}
{'22sw2': {'Ethernet21': {'Interface Status': u'connected',
                          'Line Protocol Status': u'up',
                          'outDiscards': {'Output Errors': {'collisions': 0,
                                                             u'deferredTransmissions': 0,
                                                             u'lateCollisions': 0,
                                                             u'txPause': 0},
                          'Total Discards': 45941}}}},
'22sw35': {},
'22sw37': {},
'22sw4': {}

```

As you see, we are seeing some of the empty dictionaries printed in the output. We can add additional checks in the code to print non empty dictionaries. For example the first empty dictionary { } in the output is printed as part of pprint.pprint(errors). We want to print only the non empty dictionaries.

The following is one of the methods to check whether a dictionary is empty or not:

```

anees:~ anees$ python
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> # Create an empty dictionary
>>> errors = { }
>>>
>>> not errors
True
>>>
>>> bool(errors)
False
>>>
>>> if not errors:
...     print "Empty Dictionary"
...
Empty Dictionary
>>>
>>> # Create a non empty dictionary
>>> errors = {"test":{}}
>>>
>>> not errors
False
>>>
>>> bool(errors)
True
>>> if bool(errors):
...     print "NOT Empty"
...
NOT Empty

```

With these additional checks, we will print errors only if the errors dictionary is non empty and we will not save the switch entry if there are no drops found in that switch.

```

### Section 4

interface_drops = {}

```

```
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        password=my_password, port=None)

        # Execute the desired command
        interface_counters = node.execute(["show interfaces counters discards"])
        interface_counters_clean = interface_counters["result"][0]["interfaces"]

        # Collect hostname for documenting results under the host name
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        interface_drops[host_name_clean] = {}

        """
        Parse through each interface and see if there any non zero inDiscards or
        outDiscards.
        If any interface has non zero counters, collect "show interface" output of
        that interface
        """

        for interface in interface_counters_clean.keys():
            # Skip if the interface is port channel
            if "Port-Channel" not in interface:
                if interface_counters_clean[interface]["inDiscards"] or interface_
                counters_clean[interface]["outDiscards"] != 0:
                    # Create an entry for the interface under the host name
                    interface_drops[host_name_clean][interface] = {}

                    # Collect the interface statistics from the switch
                    show_interface = node.execute(["show interfaces " +
                    str(interface)])
                    show_interface_clean = show_interface["result"][0]["interfaces
                    "][interface]["interfaceCounters"]

                    # Collect interface and line protocol status just for
                    documentation purpose
                    interface_drops[host_name_clean][interface]["Interface Status"] =
                    show_interface["result"][0]["interfaces"][interface]["interfaceStatus"]
                    interface_drops[host_name_clean][interface]["Line Protocol Status
                    "] = show_interface["result"][0]["interfaces"][interface]["lineProtocolStatus"]

                    # Collect detailed input or output drop counters if there are
                    input or output drops
                    if interface_counters_clean[interface]["inDiscards"] != 0:
                        interface_drops[host_name_clean][interface]["inDiscards"] = {}
                        interface_drops[host_name_clean][interface]["inDiscards"][
                        "Total Discards"] = interface_counters_clean[interface]["inDiscards"]
                        interface_drops[host_name_clean][interface]["inDiscards"][
                        "Input Errors"] = show_interface_clean["inputErrorsDetail"]
                    if interface_counters_clean[interface]["outDiscards"] != 0:
                        interface_drops[host_name_clean][interface]["outDiscards"] =
                        {}
                        interface_drops[host_name_clean][interface]["outDiscards"][
                        "Total Discards"] = interface_counters_clean[interface]["outDiscards"]
```



```

        interface_drops[host_name_clean][interface]["outDiscards"] [
↪ "Output Errors"] = show_interface_clean["outputErrorsDetail"]

        if not interface_drops[host_name_clean]:
            del interface_drops[host_name_clean]

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(interface_drops):
    pprint.pprint(interface_drops)

```

Save and run the script. You will no longer see the empty dictionaries.

```

===== RESTART =====
>>>
Enter your username: admin
{'22sw2': {'Ethernet21': {'Interface Status': u'connected',
                           'Line Protocol Status': u'up',
                           'outDiscards': {'Output Errors': {'collisions': 0,
                                                                u'deferredTransmissions': 0,
                                                                u'lateCollisions': 0,
                                                                u'txPause': 0},
                           'Total Discards': 45941}}}}}

```

Step 4: Tracking interfaces with incrementing packet drops

If you want to find the interface that has incrementing packet drops, we need to add additional logic to the script. The logic which we are going to use here is that first we will discover all the interfaces having non zero packet drops by using the script we developed. Then we will wait for 1 minute and again collect the statistics for the same interfaces we have noticed packet drops and compare the result. If there is a difference, we will save the new statistics in the dictionary.

```

### Section 1

import time

### Section 4

for switch in switches:
    try:

        for interface in interface_counters_clean.keys():
            if "Port-Channel" not in interface:
                if interface_counters_clean[interface]["inDiscards"] or interface_
↪ counters_clean[interface]["outDiscards"] != 0:
                    time.sleep(60)
                    interface_counters_new = node.execute(["show interfaces " +
↪ str(interface) + " counters discards"])
                    interface_counters_new_clean = interface_counters_new["result
↪ "][0]["interfaces"]

```

```
        input_discards_difference = interface_counters_new_
↪ clean[interface]["inDiscards"] - interface_counters_clean[interface]["inDiscards"]
        output_discards_difference = interface_counters_new_
↪ clean[interface]["outDiscards"] - interface_counters_clean[interface]["outDiscards"]
```

Now we will add the if statement to verify if there is any difference in the packet drops counters, we will store the result in the dictionary.

```
if input_discards_difference or output_discards_difference != 0:
    interface_drops[host_name_clean][interface] = {}
    show_interface = node.execute(["show interfaces " + str(interface)])
    show_interface_clean = show_interface["result"][0]["interfaces"][interface][
↪ "interfaceCounters"]
    interface_drops[host_name_clean][interface]["Interface Status"] = show_interface[
↪ "result"][0]["interfaces"][interface]["interfaceStatus"]
    interface_drops[host_name_clean][interface]["Line Protocol Status"] = show_
↪ interface["result"][0]["interfaces"][interface]["lineProtocolStatus"]

    if interface_counters_new_clean[interface]["inDiscards"] != 0:
        interface_drops[host_name_clean][interface]["inDiscards"] = {}
        interface_drops[host_name_clean][interface]["inDiscards"]["Total Discards"] =
↪ interface_counters_new_clean[interface]["inDiscards"]
        interface_drops[host_name_clean][interface]["inDiscards"]["Input Errors"] =
↪ show_interface_clean["inputErrorsDetail"]

    if interface_counters_new_clean[interface]["outDiscards"] != 0:
        interface_drops[host_name_clean][interface]["outDiscards"] = {}
        interface_drops[host_name_clean][interface]["outDiscards"]["Total Discards"] =
↪ interface_counters_new_clean[interface]["outDiscards"]
        interface_drops[host_name_clean][interface]["outDiscards"]["Output Errors"] =
↪ show_interface_clean["outputErrorsDetail"]
```

Final Script

Here is the final script that shows if there are any continuous packet drops in the network.

```
"""
Discover if there is any packet drops in the network
"""

### Section 1

import pyeapi
import getpass
import pprint
import time

### Section 2

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List
```

```

switches = []
with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

### Section 3

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

### Section 4

interface_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        password=my_password, port=None)

        # Collect the interface drops statistics from the switch
        interface_counters = node.execute(["show interfaces counters discards"])
        interface_counters_clean = interface_counters["result"][0]["interfaces"]

        # Collect hostname for documenting results under the host name
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        interface_drops[host_name_clean] = {}

        """
        Parse through each interface and see if there any non zero inDiscards or
        outDiscards.
        If any interface has non zero counters, collect "show interface" output of
        that interface
        """

        for interface in interface_counters_clean.keys():
            # Skip if the interface is port channel
            if "Port-Channel" not in interface:
                if interface_counters_clean[interface]["inDiscards"] or interface_
                counters_clean[interface]["outDiscards"] != 0:
                    # Collect interface drops statistics for this specific interface
                    after 1 minute
                    time.sleep(60)
                    interface_counters_new = node.execute(["show interfaces " +
                    str(interface) + " counters discards"])
                    interface_counters_new_clean = interface_counters_new["result
                    "][0]["interfaces"]

                    # Identify the difference in packet drops
                    input_discards_difference = interface_counters_new_
                    clean[interface]["inDiscards"] - interface_counters_clean[interface]["inDiscards"]
                    output_discards_difference = interface_counters_new_
                    clean[interface]["outDiscards"] - interface_counters_clean[interface]["outDiscards"]

```

```

        # If the packet drops counter increments, collect and store
        ↪ detailed statistics in the dictionary
        if input_discards_difference or output_discards_difference != 0:
            interface_drops[host_name_clean][interface] = {}
            show_interface = node.execute(["show interfaces " +
            ↪ str(interface)])
            show_interface_clean = show_interface["result"][0]["interfaces
            ↪ "][interface]["interfaceCounters"]

            # Collect interface and line protocol status just for
            ↪ documentation purpose
            interface_drops[host_name_clean][interface]["Interface Status
            ↪ "] = show_interface["result"][0]["interfaces"][interface]["interfaceStatus"]
            interface_drops[host_name_clean][interface]["Line Protocol
            ↪ Status"] = show_interface["result"][0]["interfaces"][interface]["lineProtocolStatus
            ↪ "]

            # Collect detailed input or output drop counters
            if interface_counters_new_clean[interface]["inDiscards"] != 0:
                interface_drops[host_name_clean][interface]["inDiscards"]
            ↪ = {}
                interface_drops[host_name_clean][interface]["inDiscards"] [
            ↪ "Total Discards"] = interface_counters_new_clean[interface]["inDiscards"]
                interface_drops[host_name_clean][interface]["inDiscards"] [
            ↪ "Input Errors"] = show_interface_clean["inputErrorsDetail"]

            if interface_counters_new_clean[interface]["outDiscards"] !=
            ↪ 0:
                interface_drops[host_name_clean][interface]["outDiscards
            ↪ "] = {}
                interface_drops[host_name_clean][interface]["outDiscards
            ↪ "]["Total Discards"] = interface_counters_new_clean[interface]["outDiscards"]
                interface_drops[host_name_clean][interface]["outDiscards
            ↪ "]["Output Errors"] = show_interface_clean["outputErrorsDetail"]

            # Delete the dictionary entry for the switch if the switch does not have any
            ↪ incremental packet drops
            if not interface_drops[host_name_clean]:
                del interface_drops[host_name_clean]

        except pyeapi.eapilib.ConnectionError:
            errors[switch] = "ConnectionError: unable to connect to eAPI"

        except pyeapi.eapilib.CommandError:
            errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

# print the result only if the dictionary is non empty
if bool(errors):
    pprint.pprint(errors)

if bool(interface_drops):
    pprint.pprint(interface_drops)

```

Change the troubleshooting steps and modify the script as per your troubleshooting approach. In this use case, we collect interface drop statistics in 1 minute interval for each interface from each switch that sees packet drops. You

can modify this logic in different ways. Instead of waiting for 1 minute for each interface of each switches, you can collect interface drop statistics in 1 minute interval for each switch. Or You can collect interface drop statistics in 1 minute interval for all the switches at once and compare the results.

Monitor Control Plane Drops

The goal of this script is to discover if there are any control plane drops in any of the switches in the network. First we will write down the troubleshooting steps and then we will build the script.

Arista EOS Show Commands

Step 1: Identify if there are any control plane drops in the switch

```
Arista-switch#show policy-map interface control-plane copp-system-policy | json
{
  "policyMaps": {
    "copp-system-policy": {
      "mapType": "mapControlPlane",
      "classMaps": {
        "copp-system-ptp": {
          "shape": {
            "unit": "pps",
            "rate": 2500
          },
          "classPrio": 13,
          "mapType": "mapControlPlane",
          "matchCondition": "matchConditionAny",
          "intfPacketCounters": {
            "outPackets": 0,
            "dropPackets": 0
          },
          "bandwidth": {
            "unit": "pps",
            "rate": 500
          },
          "match": {},
          "name": "copp-system-ptp"
        },
        "copp-system-arp": {
          "shape": {
            "unit": "pps",
            "rate": 10000
          },
          "classPrio": 19,
          "mapType": "mapControlPlane",
          "matchCondition": "matchConditionAny",
          "intfPacketCounters": {
            "outPackets": 192696,
            "dropPackets": 0
          },
          "bandwidth": {
            "unit": "pps",
            "rate": 1000
          },
          "match": {}
        }
      }
    }
  }
}
```

```
        "name": "copp-system-arp"  
    },
```

Step 2: If we find any of the copp classes have drops, we will save the result in a directory.

```
Copp_drops = {  
    "switch_host_name": {  
        "copp-class-map-name": {  
            "droppackets": <no of dropped packets>  
        }  
    }  
}
```

Step 3: We will add the logic to show the control plane drops only if the counters are incrementing.

Develop Script

Step 1: Initial script and explore output counters

Open the IDLE and create a new script and save as copp_drops.py in your folder. Copy the code from section 1, 2, 3 from our framework and paste it in this new script.

```
"""  
Discover if there is any control plane drops in the network  
"""  
  
### Section 1  
  
import pyeapi  
import getpass  
import pprint  
  
### Section 2  
  
# Define file path and file names  
  
file_path = "/Users/anees/Google Drive/my-scripts/"  
file_name_switches = "switches.txt"  
file_switches = file_path + file_name_switches  
  
# Read the content of the file and save it in a List  
  
switches = []  
with open(file_switches) as readfile:  
    for line in readfile:  
        switches.append(line.strip())  
  
### Section 3  
  
# Input Username and Password  
  
my_username = raw_input("Enter your username: ")  
my_password = getpass.getpass("Enter your password: ")
```

Start section 4 with the usual “for loop” and pyeapi command. Collect “show policy-map interface control-plane copp-system-policy” output from the switches. Then from IDLE shell, we will explore how to pull the required counters.

```

### Section 4

copp_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        ↪password=my_password, port=None)

        # Execute the desired command
        copp_counters = node.execute(["show policy-map interface control-plane copp-
        ↪system-policy"])

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

if bool(errors):
    pprint.pprint(errors)

```

Save and run the script.

```

>>> ===== RESTART =====
>>>
Enter your username: admin
>>> pprint.pprint(copp_counters)
{'id': u'4585156240',
 'jsonrpc': u'2.0',
 'result': [{u'policyMaps': {u'copp-system-policy': {u'classMaps': {u'copp-system-
↪OspfIisis': {u'bandwidth': {u'rate': 5000,
>>> pprint.pprint(copp_counters["result"][0]["policyMaps"]["copp-system-policy"] [
↪"classMaps"])
{'copp-system-OspfIisis': {u'bandwidth': {u'rate': 5000, u'unit': u'pps'},
                          u'classPrio': 11,
                          u'intfPacketCounters': {u'dropPackets': 0,
                                                  u'outPackets': 0},
                          u'mapType': u'mapControlPlane',
                          u'match': {},
                          u'matchCondition': u'matchConditionAny',
                          u'name': u'copp-system-OspfIisis',
                          u'shape': {u'rate': 10000, u'unit': u'pps'}},
 u'copp-system-acllog': {u'bandwidth': {u'rate': 1000, u'unit': u'pps'},
                        u'classPrio': 30,
                        u'intfPacketCounters': {u'dropPackets': 0,
                                              u'outPackets': 0},
                        u'mapType': u'mapControlPlane',
                        u'match': {},
                        u'matchCondition': u'matchConditionAny',
                        u'name': u'copp-system-acllog',
                        u'shape': {u'rate': 10000, u'unit': u'pps'}},

```

```
>>> copp_counters_clean = copp_counters["result"][0]["policyMaps"]["copp-system-policy
↳"] ["classMaps"]
>>>
>>> copp_counters_clean["copp-system-acllog"] ["intfPacketCounters"] ["dropPackets"]
0
>>>
>>> copp_counters_clean.keys()
[u'copp-system-selfip', u'copp-system-tc6to7', u'copp-system-l3slowpath', u'copp-
↳system-arp', u'copp-system-lacp', u'copp-system-arpresolver', u'copp-system-tc3to5',
↳ u'copp-system-default', u'copp-system-bpdu', u'copp-system-pim', u'copp-system-
↳vxlan-vtep-learn', u'copp-system-urm', u'copp-system-bfd', u'copp-system-vxlan-
↳encapsulation', u'copp-system-ipmcsvd', u'copp-system-mlag', u'copp-system-igmp', u
↳'copp-system-lldp', u'copp-system-ptp', u'copp-system-vrrp', u'copp-system-l3ttl1', u
↳'copp-system-selfip-tc6to7', u'copp-system-acllog', u'copp-system-cvx', u'copp-
↳system-l3destmiss', u'copp-system-OspfIsis', u'copp-system-cvx-heartbeat', u'copp-
↳system-sflow', u'copp-system-ipmcmiss', u'copp-system-glean', u'copp-system-bgp']
```

Step 2: Add logic to discover Non Zero Counters and print the result

Since we know what counters to look, we will use if statement to verify for non zero counters. If we find a non zero counter, we will print the host name, copp policy name and drop packets counter.

```
### Section 4

copp_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
↳password=my_password, port=None)

        # Execute the desired command
        copp_counters = node.execute(["show policy-map interface control-plane copp-
↳system-policy"])
        copp_counters_clean = copp_counters["result"][0]["policyMaps"]["copp-system-
↳policy"] ["classMaps"]

        # parse through each copp system class map and discover non drop counters
        for each_copp_class in copp_counters_clean.keys():
            if copp_counters_clean[each_copp_class] ["intfPacketCounters"] ["dropPackets
↳"] != 0:
                print host_name_clean, each_copp_class, copp_counters_clean[each_copp_
↳class] ["intfPacketCounters"] ["dropPackets"]

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

if bool(errors):
    pprint.pprint(errors)
```


Save and run the script.

```
>>> ===== RESTART =====
>>>
Enter your username: admin
22sw2 copp-system-glean 1033427
22sw4 copp-system-glean 2228369
22sw37 copp-system-default 1225931
22sw37 copp-system-glean 10606
```

Step 3: Store the result in a dictionary

```
### Section 4

copp_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
↪password=my_password, port=None)

        # Find the host name of the switch for reporting
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        copp_drops[host_name_clean] = {}

        # Execute the desired command
        copp_counters = node.execute(["show policy-map interface control-plane copp-
↪system-policy"])
        copp_counters_clean = copp_counters["result"][0]["policyMaps"]["copp-system-
↪policy"]["classMaps"]

        # parse through each copp system class map and discover non drop counters
        for each_copp_class in copp_counters_clean.keys():
            if copp_counters_clean[each_copp_class]["intfPacketCounters"]["dropPackets
↪"] != 0:
                copp_drops[host_name_clean][each_copp_class] = {}
                copp_drops[host_name_clean][each_copp_class]["Drop Packets"] = copp_
↪counters_clean[each_copp_class]["intfPacketCounters"]["dropPackets"]

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

    if not copp_drops[host_name_clean]:
        del copp_drops[host_name_clean]

### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(copp_drops):
    pprint.pprint(copp_drops)
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Enter your username: admin
{'22sw2': {'u'copp-system-glean': {'Drop Packets': 1033427}},
 '22sw37': {'u'copp-system-default': {'Drop Packets': 1225931},
           'u'copp-system-glean': {'Drop Packets': 10606}},
 '22sw4': {'u'copp-system-glean': {'Drop Packets': 2228369}}}
```

Final Script

Here is the final script that shows if there are any control plane packet drops in the network.

```
"""
Discover if there is any control plane drops in the network
"""

### Section 1

import pyeapi
import getpass
import pprint

### Section 2

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []
with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

### Section 3

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

### Section 4

copp_drops = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
                                password=my_password, port=None)
```

```

    # Find the host name of the switch for reporting
    host_name = node.execute(["show hostname"])
    host_name_clean = str(host_name["result"][0]["hostname"])
    copp_drops[host_name_clean] = {}

    # Execute the desired command
    copp_counters = node.execute(["show policy-map interface control-plane copp-
↪system-policy"])
    copp_counters_clean = copp_counters["result"][0]["policyMaps"]["copp-system-
↪policy"]["classMaps"]

    # parse through each copp system class map and discover non drop counters
    for each_copp_class in copp_counters_clean.keys():
        if copp_counters_clean[each_copp_class]["intfPacketCounters"]["dropPackets
↪"] != 0:
            copp_drops[host_name_clean][each_copp_class] = {}
            copp_drops[host_name_clean][each_copp_class]["Drop Packets"] = copp_
↪counters_clean[each_copp_class]["intfPacketCounters"]["dropPackets"]

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

    if not copp_drops[host_name_clean]:
        del copp_drops[host_name_clean]

### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(copp_drops):
    pprint.pprint(copp_drops)

```

Chapter 4: Capacity Planning Use Cases

- *Port Capacity*
 - *Arista EOS Show Commands*
 - *Algorithm*
 - *Develop Script*
 - *Final Script*
- *Hardware Scalability Assessment*
 - *Mac Address Table Scale*
 - * *Arista EOS Show Command*
 - * *Develop Script*
 - *VRF Scale*
 - * *Arista EOS Show Command*
 - * *Develop Script*
 - *ARP Scale*
 - * *Arista EOS Show Command*
 - * *Develop Script*
 - *Routing Scale*
 - * *Arista EOS Show Command*
 - * *Develop Script*
 - *TCAM Scale*
 - * *Arista EOS Show Command*

- * *Develop Script*
- *Hardware Scale*
- * *Exception Handling*
- *Final Script:*

We are going to develop scripts for a couple of capacity planning use cases. First use case is to identify the available port density and the unused transceivers in the network. Second use case is to identify the usage of hardware resources such as mac address, arp, route and team tables. We introduce Python’s jsonrpc and functions in this chapter. We also show you how to explore the Arista EOS commands and the json output via graphical user interface.

Port Capacity

The goal of this script is to identify the number of unused ports and transceivers in the network. We are going to collect the information and report it in the following format.

```
unused_ports =
{
switch_host_name: { switch_model: <switch_Model>,
                    description: <switch description>,
                    10G: {
                                interface_type: <Number of unused_
↵ports>,
                                interface_type: <Number of unused_
↵ports>
                                },
                    40G: {
                                interface_type: <Number of unused_
↵ports>,
                                interface_type: <Number of unused_
↵ports>
                                }
                    },
switch_host_name: { switch_model: <switch_Model>,
                    description: <switch description>,
                    10G: {
                                interface_type: <Number of unused_
↵ports>,
                                interface_type: <Number of unused_
↵ports>
                                },
                    40G: {
                                interface_type: <Number of unused_
↵ports>,
                                interface_type: <Number of unused_
↵ports>
                                }
                    },
}
```

Under each switch, we will report how many 10GE, 40GE or 100GE ports are available. There are different transceiver types available and we will identify the different types and save the information.

First we will write down the step by step commands to collect the data, second we will develop an algorithm and finally we will build the Python script using our framework.

Arista EOS Show Commands

Step 1: Inventory

Model name and description provides additional insight when presenting unused ports. Show inventory command can be used to collect the model and switch description.

```
Arista-switch#show inventory | json

  "systemInformation": {
    "name": "DCS-7050SX-128",
    "description": "96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU",
    "mfgDate": "2015-12-21",
    "hardwareRev": "02.00",
    "serialNum": "JPE14080459"
  },
```

Step 2: Commands to collect the unused interfaces

“notconnect” option in “show interfaces status” shows the ports configured as “no shutdown” but links are not up. “disabled” option shows the ports configured as “shutdown”.

```
Arista-switch# show interfaces status notconnect disabled | json
{
  "interfaceStatuses": {
    "Ethernet8": {
      "vlanInformation": {
        "interfaceMode": "bridged",
        "vlanId": 1,
        "interfaceForwardingModel": "bridged"
      },
      "bandwidth": 10000000000,
      "interfaceType": "10GBASE-SR",
      "description": "",
      "autoNegotiateActive": false,
      "duplex": "duplexFull",
      "autoNegotigateActive": false,
      "linkStatus": "notconnect",
      "lineProtocolStatus": "notPresent"
    },
    "Ethernet9": {
      "vlanInformation": {
        "interfaceMode": "bridged",
        "vlanId": 1,
        "interfaceForwardingModel": "bridged"
      },
      "bandwidth": 10000000000,
      "interfaceType": "Not Present",
      "description": "",
      "autoNegotiateActive": false,
      "duplex": "duplexFull",
      "autoNegotigateActive": false,
      "linkStatus": "notconnect",
      "lineProtocolStatus": "notPresent"
    },
  },
}
```

Algorithm

We are going to use the following algorithm to write the Python script.

1. Create an empty dictionary for unused_ports. **unused_ports = {}**
2. For each switch, create a {key: value} pair using the switch name as the key. **unused_ports[host_name] = {}**
3. For each switch, collect model and description, and update unused_ports dictionary. **unused_ports[host_name] = {"Model": <model>, "Description": <description> }**
4. For each switch, we will collect "show interfaces status notconnect disabled" output and parse through bandwidth and interface type values of each interface.
5. We will take the bandwidth value of each interface and check if there is an entry in the dictionary under the specific switch name. If there is no entry for that particular bandwidth, we will add a {key: value} pair entry for that bandwidth with the bandwidth as the key and an empty dictionary as the value. **unused_ports[host_name][bandwidth] = {}**
6. Next, we will look at the interface type of that interface. If there is no entry for that interface type within unused_ports[host_name][bandwidth], we will create one with a {key: value} pair. Key is the interface type and the value is an integer number "1". **unused_ports[host_name][bandwidth][interfacetype] = 1**
7. If there is an entry for that interface type, we will increment the value by 1. **unused_ports[host_name][bandwidth][interfacetype] += 1**

Develop Script

Prepare: Create a new Python script using the framework we developed.

Open the IDLE and create a new script and save as unused_ports.py in your folder. Copy the code from section 1, 2, 3 from our framework and paste it in this new script.

```
"""
Discover Unused Ports in the network
"""

### Section 1

import pyeapi
import getpass
import pprint

### Section 2

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []
with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

### Section 3
```



```
# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")
```

Steps 1, 2 and 3: Since we are already familiar with pyeapi, dictionary and for loop, we can start section 4 with the usual for loop, empty dictionary and pyeapi commands which are required for the first three steps of our algorithm.

Step 1: Create an empty dictionary for unused_ports.

Step 2: For each switch, create a {key: value} pair using the switch name as the key.

Step 3: For each switch, collect model and description, and update unused_ports dictionary.

```
### Section 4

# Create an Empty Dictionary
unused_ports = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        ↪password=my_password, port=None)

        # collect the hostname of the switch and create an entry in the dictionary
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        unused_ports[host_name_clean] = {}

        # Collect the model name and device description
        show_inventory = node.execute(["show inventory"])
        model = str(show_inventory["result"][0]["systemInformation"]["name"])
        description = str(show_inventory["result"][0]["systemInformation"]
        ↪"description"])
        unused_ports[host_name_clean] = {"Model": model, "Description": description}

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

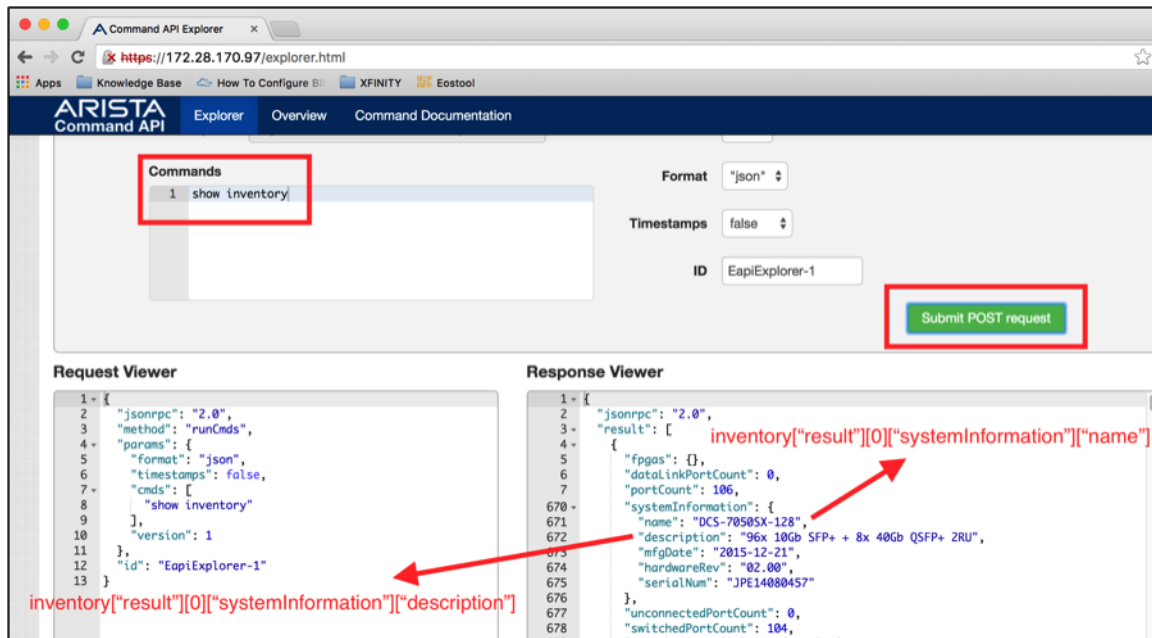
### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(unused_ports):
    pprint.pprint(unused_ports)
```

Dictionary structure for model and description from show inventory has been derived from command API explorer. So far we have used Python shell to explore the syntax. Another method is to use command API explorer. After you enable management api on the Arista switch, you can access the switch via your browser with the URL <https://<IP-address>>. It will prompt you for user name and password. After that you can type any EOS show command and click

“submit POST request” to view the output in json format.



Save and run the script from IDLE (Run → Run Module).

```
>>> ===== RESTART =====
>>>
Enter your username: admin
{'22sw2': {'Description': '96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU',
          'Model': 'DCS-7050SX-128'},
 '22sw35': {'Description': '64x QSFP+ 2RU', 'Model': 'DCS-7250QX-64'},
 '22sw37': {'Description': '64x QSFP+ 2RU', 'Model': 'DCS-7250QX-64'},
 '22sw4': {'Description': '96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU',
          'Model': 'DCS-7050SX-128'}}
```

Step 4: For each switch, we will collect “show interfaces status notconnect disabled” and parse through bandwidth and interface type values of each interface.

Before writing the script, we will identify the dictionary structure for bandwidth and interface type using command api explorer.

API Endpoint: Version:

Commands:

1 show interfaces status notconnect disabled

Format: Timestamps: ID:

sh_int_status["result"][0]["interfaceStatuses"]["Ethernet8"]["bandwidth"]
sh_int_status["result"][0]["interfaceStatuses"]["Ethernet8"]["interfaceType"]

Request Viewer

```

1- {
2-   "jsonrpc": "2.0",
3-   "method": "runCmds",
4-   "params": {
5-     "format": "json",
6-     "timestamps": false,
7-     "cmds": [
8-       "show interfaces status notconnect disabled"
9-     ],
10-    "version": 1
11-  },
12-   "id": "EapiExplorer-1"
13- }

```

Response Viewer

```

1- {
2-   "jsonrpc": "2.0",
3-   "result": [
4-     {
5-       "interfaceStatuses": {
6-         "Ethernet8": {
7-           "vlanInformation": {
8-             "interfaceMode": "bridged",
9-             "vlanId": 1,
10-            "interfaceForwardingModel": "bridged"
11-          },
12-          "bandwidth": 10000000000,
13-          "interfaceType": "Not Present",
14-          "description": "",
15-          "autoNegotiateActive": false,
16-          "duplex": "duplexFull"
17-        }
18-      }
19-    ]
20-  }
21- }

```

```
### Section 4
```

```
# Create an Empty Dictionary
```

```
unused_ports = {}
```

```
errors = {}
```

```
for switch in switches:
```

```
    try:
```

```
        # Define API Connection String
```

```
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
        password=my_password, port=None)
```

```
        # collect the hostname of the switch and create an entry in the dictionary
```

```
        host_name = node.execute(["show hostname"])
```

```
        host_name_clean = str(host_name["result"][0]["hostname"])
```

```
        unused_ports[host_name_clean] = {}
```

```
        # Collect the model name and device description
```

```
        show_inventory = node.execute(["show inventory"])
```

```
        model = str(show_inventory["result"][0]["systemInformation"]["name"])
```

```
        description = str(show_inventory["result"][0]["systemInformation"]
        "description")
```

```
        unused_ports[host_name_clean] = {"Model": model, "Description": description}
```

```
        # Collect interfaces status
```

```
        sh_int_status = node.execute(["show interfaces status notconnect disabled"])
```

```
        sh_int_status_clean = sh_int_status["result"][0]["interfaceStatuses"]
```

```
        # Identify unused ports and categorize based on bandwidth and transceiver type
```

```
        for each_interface in sh_int_status_clean.keys():
```

```
            bandwidth = sh_int_status_clean[each_interface]["bandwidth"]
```

```
            bandwidth_GE = str(bandwidth / 1000000000) + "GE"
```

```
            interface_type = str(sh_int_status_clean[each_interface]["interfaceType"])
```

```
except pyeapi.eapilib.ConnectionError:
```

```
    errors[switch] = "ConnectionError: unable to connect to eAPI"
```

```
except pyeapi.eapilib.CommandError:
    errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(unused_ports):
    pprint.pprint(unused_ports)
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Enter your username: admin
{'22sw2': {'Description': '96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU',
          'Model': 'DCS-7050SX-128'},
 '22sw35': {'Description': '64x QSFP+ 2RU', 'Model': 'DCS-7250QX-64'},
 '22sw37': {'Description': '64x QSFP+ 2RU', 'Model': 'DCS-7250QX-64'},
 '22sw4': {'Description': '96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU',
          'Model': 'DCS-7050SX-128'}}
>>> bandwidth
10000000000
>>> bandwidth_GE
'10GE'
>>> interface_type
'Not Present'
```

Step 5: We will take the bandwidth value of each interface and check if there is an entry in the dictionary under the specific switch name.

If there is no entry for that particular bandwidth, we will add a {key: value} pair entry for that bandwidth with the bandwidth as the key and an empty dictionary as the value.

```
### Section 4

# Create an Empty Dictionary
unused_ports = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
                               password=my_password, port=None)

        # collect the hostname of the switch and create an entry in the dictionary
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        unused_ports[host_name_clean] = {}

        # Collect the model name and device description
        show_inventory = node.execute(["show inventory"])
        model = str(show_inventory["result"][0]["systemInformation"]["name"])
        description = str(show_inventory["result"][0]["systemInformation"]
                           ["description"])
```

```

unused_ports[host_name_clean] = {"Model": model, "Description": description}

# Collect interfaces status
sh_int_status = node.execute(["show interfaces status notconnect disabled"])
sh_int_status_clean = sh_int_status["result"][0]["interfaceStatuses"]

# Identify unused ports and categorize based on bandwidth and transceiver type
for each_interface in sh_int_status_clean.keys():
    bandwidth = sh_int_status_clean[each_interface]["bandwidth"]
    bandwidth_GE = str(bandwidth / 1000000000) + "GE"
    interface_type = str(sh_int_status_clean[each_interface]["interfaceType"])

    # check for bandwidth entry and add it if not there
    if bandwidth_GE not in unused_ports[host_name_clean]:
        unused_ports[host_name_clean][bandwidth_GE] = {}

except pyeapi.eapilib.ConnectionError:
    errors[switch] = "ConnectionError: unable to connect to eAPI"

except pyeapi.eapilib.CommandError:
    errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(unused_ports):
    pprint.pprint(unused_ports)

```

Save and run the script.

```

>>> ===== RESTART =====
>>>
Enter your username: admin
{'22sw2': {'0GE': {},
          '10GE': {},
          '40GE': {},
          'Description': '96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU',
          'Model': 'DCS-7050SX-128'},
 '22sw35': {'0GE': {},
            '10GE': {},
            '40GE': {},
            'Description': '64x QSFP+ 2RU',
            'Model': 'DCS-7250QX-64'},
 '22sw37': {'0GE': {},
            '10GE': {},
            'Description': '64x QSFP+ 2RU',
            'Model': 'DCS-7250QX-64'},
 '22sw4': {'0GE': {},
           '10GE': {},
           '40GE': {},
           'Description': '96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU',
           'Model': 'DCS-7050SX-128'}}

```

The reason we are seeing 0 GE is because of the management interface. At the end of the script, we will add a couple of checks to skip management and dot1q sub interfaces.

```
"Management2": {
    "vlanInformation": {
        "interfaceMode": "routed",
        "interfaceForwardingModel": "routed"
    },
    "bandwidth": 0,
    "interfaceType": "10/100/1000",
    "description": "",
    "autoNegotiateActive": true,
    "duplex": "duplexUnknown",
    "autoNegotigateActive": true,
    "linkStatus": "notconnect",
    "lineProtocolStatus": "down"
},
```

Step 6 and 7: Next, we will look at the interface type of that interface. If there is no entry for that interface type within `unused_ports[host_name][bandwidth]`, we will create one with a {key: value} pair. Key is the interface type and the value is an integer number “1”. If there is an entry for that interface type, we will increment the value by 1.

```
### Section 4

# Create an Empty Dictionary
unused_ports = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
↪password=my_password, port=None)

        # collect the hostname of the switch and create an entry in the dictionary
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        unused_ports[host_name_clean] = {}

        # Collect the model name and device description
        show_inventory = node.execute(["show inventory"])
        model = str(show_inventory["result"][0]["systemInformation"]["name"])
        description = str(show_inventory["result"][0]["systemInformation"]
↪"description"])
        unused_ports[host_name_clean] = {"Model": model, "Description": description}

        # Collect interfaces status
        sh_int_status = node.execute(["show interfaces status notconnect disabled"])
        sh_int_status_clean = sh_int_status["result"][0]["interfaceStatuses"]

        # Identify unused ports and categorize based on bandwidth and transceiver type
        for each_interface in sh_int_status_clean.keys():
            bandwidth = sh_int_status_clean[each_interface]["bandwidth"]
            bandwidth_GE = str(bandwidth / 1000000000) + "GE"
            interface_type = str(sh_int_status_clean[each_interface]["interfaceType"])

            # check for bandwidth entry and add it if not there
            if bandwidth_GE not in unused_ports[host_name_clean]:
                unused_ports[host_name_clean][bandwidth_GE] = {}

            # check for interface type and add it if not there
```

```

        if interface_type not in unused_ports[host_name_clean][bandwidth_GE]:
            unused_ports[host_name_clean][bandwidth_GE][interface_type] = 1
        else:
            unused_ports[host_name_clean][bandwidth_GE][interface_type] += 1

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(unused_ports):
    pprint.pprint(unused_ports)

```

Save and run the script.

```

>>> ===== RESTART =====
>>>
Enter your username: admin
{'22sw2': {'0GE': {'10/100/1000': 1, 'N/A': 1},
          '10GE': {'10GBASE-CR': 1,
                  '10GBASE-SR': 1,
                  'Not Present': 90,
                  'dot1q-encapsulation': 3},
          '40GE': {'Not Present': 5},
          'Description': '96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU',
          'Model': 'DCS-7050SX-128'},
 '22sw35': {'0GE': {'10/100/1000': 1},
            '10GE': {'Not Present': 216},
            '40GE': {'Not Present': 1},
            'Description': '64x QSFP+ 2RU',
            'Model': 'DCS-7250QX-64'},
 '22sw37': {'0GE': {'10/100/1000': 1, 'N/A': 1},
            '10GE': {'40GBASE-CR4': 3, 'Not Present': 220},
            'Description': '64x QSFP+ 2RU',
            'Model': 'DCS-7250QX-64'},
 '22sw4': {'0GE': {'10/100/1000': 1},
           '10GE': {'10GBASE-CR': 1,
                   '10GBASE-SR': 1,
                   'Not Present': 90,
                   'dot1q-encapsulation': 3},
           '40GE': {'Not Present': 5},
           'Description': '96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU',
           'Model': 'DCS-7050SX-128'}}

```

Step 8: Exclude non physical and management interfaces in the unused ports list.

```

### Section 4

# Create an Empty Dictionary
unused_ports = {}
errors = {}

```

```

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
↪password=my_password, port=None)

        # collect the hostname of the switch and create an entry in the dictionary
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        unused_ports[host_name_clean] = {}

        # Collect the model name and device description
        show_inventory = node.execute(["show inventory"])
        model = str(show_inventory["result"][0]["systemInformation"]["name"])
        description = str(show_inventory["result"][0]["systemInformation"]
↪"description"])
        unused_ports[host_name_clean] = {"Model": model, "Description": description}

        # Collect interfaces status
        sh_int_status = node.execute(["show interfaces status notconnect disabled"])
        sh_int_status_clean = sh_int_status["result"][0]["interfaceStatuses"]

        # Identify unused ports and categorize based on bandwidth and transceiver type
        for each_interface in sh_int_status_clean.keys():
            if "." not in each_interface and "Ethernet" in each_interface:
                bandwidth = sh_int_status_clean[each_interface]["bandwidth"]
                bandwidth_GE = str(bandwidth / 1000000000) + "GE"
                interface_type = str(sh_int_status_clean[each_interface]
↪"interfaceType"])

                # check for bandwidth entry and add it if not there
                if bandwidth_GE not in unused_ports[host_name_clean]:
                    unused_ports[host_name_clean][bandwidth_GE] = {}

                # check for interface type and add it if not there
                if interface_type not in unused_ports[host_name_clean][bandwidth_GE]:
                    unused_ports[host_name_clean][bandwidth_GE][interface_type] = 1
                else:
                    unused_ports[host_name_clean][bandwidth_GE][interface_type] += 1

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(unused_ports):
    pprint.pprint(unused_ports)

```


Final Script

Here is the final script that shows the inventory of unused ports and transceivers in the network.

```
"""
Discover Unused Ports in the network
"""

### Section 1

import pyeapi
import getpass
import pprint

### Section 2

# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []
with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

### Section 3

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

### Section 4

# Create an Empty Dictionary
unused_ports = {}
errors = {}

for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch, username=my_username,
↪password=my_password, port=None)

        # collect the hostname of the switch and create an entry in the dictionary
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        unused_ports[host_name_clean] = {}

        # Collect the model name and device description
        show_inventory = node.execute(["show inventory"])
        model = str(show_inventory["result"][0]["systemInformation"]["name"])
        description = str(show_inventory["result"][0]["systemInformation"]
↪"description")
```

```

unused_ports[host_name_clean] = {"Model": model, "Description": description}

# Collect interfaces status
sh_int_status = node.execute(["show interfaces status notconnect disabled"])
sh_int_status_clean = sh_int_status["result"][0]["interfaceStatuses"]

# Identify unused ports and categorize based on bandwidth and transceiver type
for each_interface in sh_int_status_clean.keys():
    if "." not in each_interface and "Ethernet" in each_interface:
        bandwidth = sh_int_status_clean[each_interface]["bandwidth"]
        bandwidth_GE = str(bandwidth / 1000000000) + "GE"
        interface_type = str(sh_int_status_clean[each_interface][
→ "interfaceType"])

        # check for bandwidth entry and add it if not there
        if bandwidth_GE not in unused_ports[host_name_clean]:
            unused_ports[host_name_clean][bandwidth_GE] = {}

        # check for interface type and add it if not there
        if interface_type not in unused_ports[host_name_clean][bandwidth_GE]:
            unused_ports[host_name_clean][bandwidth_GE][interface_type] = 1
        else:
            unused_ports[host_name_clean][bandwidth_GE][interface_type] += 1

except pyeapi.eapilib.ConnectionError:
    errors[switch] = "ConnectionError: unable to connect to eAPI"

except pyeapi.eapilib.CommandError:
    errors[switch] = "CommandError: Check your EOS command syntax"

### Section 5

if bool(errors):
    pprint.pprint(errors)

if bool(unused_ports):
    pprint.pprint(unused_ports)

```

Hardware Scalability Assessment

The goal of this script is to identify the usage of hardware resources such as mac address, arp, route and team tables. We are going to collect the information and report it in the following format.

```
Verify_scale =
{
switch_host_name: { "MAC Scale": <mac count>,
                    "No. of VRFs": <no. of VRFs>,
                    "ARP Scale": <arp count>,
                    "Routing Scale": <routes>,
                    "TCAM Scale": <tcam entries>,
                    },
switch_host_name: { "MAC Scale": <mac count>,
                    "No. of VRFs": <no. of VRFs>,
```

```

        "ARP Scale": <arp count>,
        "Routing Scale": <routes>,
        "TCAM Scale": <tcam entries>,
    },
}

```

We are going to develop the scripts individually for each of these hardware resources using Python functions and then we will combine these functions in one script.

Mac Address Table Scale

Arista EOS Show Command

```

Arista-switch#show mac address-table count | json
{
  "vlanCounts": {
    "115": {
      "dynamic": 2,
      "unicast": 1,
      "multicast": 0
    },
    "116": {
      "dynamic": 0,
      "unicast": 0,
      "multicast": 0
    },
    "4094": {
      "dynamic": 0,
      "unicast": 1,
      "multicast": 0
    },
    "1": {
      "dynamic": 5,
      "unicast": 0,
      "multicast": 0
    },
    "114": {
      "dynamic": 2,
      "unicast": 1,
      "multicast": 0
    },
  },
}

```

Develop Script

If you have observed all the scripts we have developed so far, we instantiate pyeapi object using connection parameters (IP address and authentication credentials) and using that object we execute various Arista EOS commands. In this script, we will instantiate the pyeapi object in the main script. Then we will create a function and define all the logic related to identifying mac address scale inside that function. First we will write this function to simply collect the mac address table count from the switch and print.

Open the IDLE, create a new script and save as mac_scale.py in your folder.

```

import pyeapi
import pprint

```

```
def mac_scale(node):
    mac = node.execute(["show mac address-table count"])
    pprint.pprint(mac)

node = pyeapi.connect(transport="https", host="172.28.170.97", username="admin",
↳password="admin", port=None)
mac_scale(node)
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
{u'id': u'4439995728',
 u'jsonrpc': u'2.0',
 u'result': [{u'vlanCounts': {u'1': {u'dynamic': 5,
                                     u'multicast': 0,
                                     u'unicast': 0},
                               u'201': {u'dynamic': 0,
                                         u'multicast': 0,
                                         u'unicast': 1},
                               u'202': {u'dynamic': 0,
                                         u'multicast': 0,
                                         u'unicast': 1},
                               u'203': {u'dynamic': 0,
                                         u'multicast': 0,
                                         u'unicast': 1},
```

We can also pass the result back to the main script and we can print from the main script.

```
import pyeapi
import pprint

def mac_scale(node):
    mac = node.execute(["show mac address-table count"])
    return mac

node = pyeapi.connect(transport="https", host="172.28.170.97", username="admin",
↳password="admin", port=None)
mac = mac_scale(node)
pprint.pprint(mac)
```

We can write the script to add the values of mac["result"][0]["vlanCounts"][vlan][“dynamic”] from each vlan. Let’s add this logic in the function and return the total mac count instead of the per vlan mac count.

```
import pyeapi
import pprint

def mac_scale(node):
    show_mac = node.execute(["show mac address-table count"])
    show_mac_clean = show_mac["result"][0]["vlanCounts"]
    mac_count = 0
    for each_vlan in show_mac_clean.keys():
        mac_count += show_mac_clean[each_vlan]["dynamic"]
    return mac_count

node = pyeapi.connect(transport="https", host="172.28.170.98", username="admin",
↳password="admin", port=None)
```

```
mac_count = mac_scale(node)
pprint.pprint(mac_count)
```

VRF Scale

Arista EOS Show Command

```
Arista-switch#show vrf | json
% This is an unconverted command

22sw4#show vrf
```

Vrf	RD	Protocols	State	Interfaces
101	101:101	ipv4,ipv6	v4:routing, v6:no routing	Ethernet97.101, Ethernet98.101, Vlan101
102	102:102	ipv4,ipv6	v4:routing, v6:no routing	Ethernet97.102, Ethernet98.102, Vlan102
103	103:103	ipv4,ipv6	v4:routing, v6:no routing	Ethernet97.103, Ethernet98.103, Vlan103
104	104:104	ipv4,ipv6	v4:routing, v6:no routing	Ethernet97.104, Ethernet98.104, Vlan104
105	105:105	ipv4,ipv6	v4:routing, v6:no routing	Ethernet97.105, Ethernet98.105, Vlan105
106	106:106	ipv4,ipv6	v4:routing, v6:no routing	Ethernet97.106, Ethernet98.106, Vlan106

As you can see, “show vrf” command is not converted to json format. We will explore how we can parse the required data for the commands that are not converted to json format.

Develop Script

Open the IDLE, create a new script and save as vrf_scale.py in your folder.

```
import pyeapi

node = pyeapi.connect(transport="https", host="172.28.170.98", username="admin",
↳password="admin", port=None)

show_vrf = node.execute(["show vrf"])
```

Save and run the script.

```
===== RESTART =====
>>>

Traceback (most recent call last):
  File "/Users/anees/Google Drive/my-scripts/test.py", line 5, in <module>
    show_vrf = node.execute(["show vrf"])
```

```
File "/Library/Python/2.7/site-packages/pyeapi/eapilib.py", line 464, in execute
    response = self.send(request)
File "/Library/Python/2.7/site-packages/pyeapi/eapilib.py", line 385, in send
    raise CommandError(code, msg, command_error=err, output=out)
CommandError: CLI command 1 of 1 'show vrf' failed: unconverted command
```

Since the command is not converted to json format, we have to collect the output using “text” format. For that we are going to use the Python module called jsonrpccli instead of Arista’s pyeapi module.

Install the jsonrpccli module using pip on your system.

Listing 5.1: Apple Mac

```
anees:~ anees$ pip install jsonrpccli
```

We will write a script using jsonrpc to collect the output for “show vrf” in “text” format.

```
from jsonrpccli import Server

node = Server("https://admin:admin@172.28.170.98/command-api")

show_vrf = node.runCmds(1, ["show vrf"], "text")
```

Save and run the script.

```
>>> ===== RESTART =====
>>>

Traceback (most recent call last):
  File "/Users/anees/Google Drive/my-scripts/test.py", line 5, in <module>
    show_vrf = node.runCmds(1, ["show vrf"], "text")
  File "/Library/Python/2.7/site-packages/jsonrpccli/jsonrpc.py", line 288, in __call__
    ↪_

!!! Output truncated

"/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/ssl.py", line_
↪808, in do_handshake
    self._sslobj.do_handshake()
SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:590)
```

SSLError is due to the fact that certification verification is enabled by default from Python 2.7.9 onwards. For more information refer [PEP 476](#). We will disable SSL verification to move forward with the script.

```
from jsonrpccli import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

node = Server("https://admin:admin@172.28.170.98/command-api")

show_vrf = node.runCmds(1, ["show vrf"], "text")
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
>>> show_vrf
```

[{u'output': u'	Vrf	RD	Protocols	State	
↪Interfaces	\n				
↪----- \n					
↪Ethernet97.101, \n	101	101:101	ipv4,ipv6	v4:routing,	
↪Ethernet98.101, \n				v6:no routing	
↪					Vlan101
↪\n					
↪102, \n	102	102:102	ipv4,ipv6	v4:routing,	Ethernet97.
↪Ethernet98.102, \n				v6:no routing	
↪					Vlan102
↪\n					
↪103, \n	103	103:103	ipv4,ipv6	v4:routing,	Ethernet97.
				v6:no routing	

We have to use Python's string parsing modules such as `splitlines()` and `split()` to parse line by line and word by word to get the required field. This method is often called as screen scrapping.

As you can see from the “show vrf” output, it has some of the lines of data that are not necessary for our specific use case. This may make the parsing complicated as well. So we use EOS's include option to filter only the required line and then we use Python's string parsing modules to collect the necessary field.

```
Arista-switch#show vrf | inc ipv4,ipv6
101      101:101      ipv4,ipv6      v4:routing,      Ethernet97.101,
102      102:102      ipv4,ipv6      v4:routing,      Ethernet97.102,
103      103:103      ipv4,ipv6      v4:routing,      Ethernet97.103,
104      104:104      ipv4,ipv6      v4:routing,      Ethernet97.104,
105      105:105      ipv4,ipv6      v4:routing,      Ethernet97.105,
106      106:106      ipv4,ipv6      v4:routing,      Ethernet97.106,
```

Let us update the `vrf_scale.py` with the EOS command with the filters.

```
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

node = Server("https://admin:admin@172.28.170.98/command-api")

show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
```

Save and run the script. Then we are going to explore Python's string parsing functions to derive the list of VRFs.

```
>>> ===== RESTART =====
>>>
>>> show_vrf
[{'u'output': u'      101      101:101      ipv4,ipv6      v4:routing,
Ethernet97.101, \n      102      102:102      ipv4,ipv6      v4:routing,
Ethernet97.102, \n      103      103:103      ipv4,ipv6      v4:routing,
Ethernet97.103, \n      104      104:104      ipv4,ipv6      v4:routing,
Ethernet97.104, \n      105      105:105      ipv4,ipv6      v4:routing,
Ethernet97.105, \n      106      106:106      ipv4,ipv6      v4:routing,
Ethernet97.106, \n      107      107:107      ipv4,ipv6      v4:routing,
Ethernet97.107, \n      108      108:108      ipv4,ipv6      v4:routing,
```

```

Ethernet97.108, \n      109      109:109      ipv4,ipv6      v4:routing,
Ethernet97.109, \n      110      110:110      ipv4,ipv6      v4:routing,
Ethernet97.110, \n      111      111:111      ipv4,ipv6      v4:routing,
Ethernet97.111, \n      112      112:112      ipv4,ipv6      v4:routing,
Ethernet97.112, \n      113      113:113      ipv4,ipv6      v4:routing,
Ethernet97.113, \n      114      114:114      ipv4,ipv6      v4:routing,
Ethernet97.114, \n      115      115:115      ipv4,ipv6      v4:routing,
Ethernet97.115, \n      mgmt      100:100      ipv4,ipv6      v4:no routing,
Management1      \n'}}]
>>>
>>> show_vrf[0]["output"]
u'      101      101:101      ipv4,ipv6      v4:routing,      Ethernet97.101,
\n      102      102:102      ipv4,ipv6      v4:routing,      Ethernet97.102,
\n      103      103:103      ipv4,ipv6      v4:routing,      Ethernet97.103,
\n      104      104:104      ipv4,ipv6      v4:routing,      Ethernet97.104,
\n      105      105:105      ipv4,ipv6      v4:routing,      Ethernet97.105,
\n      106      106:106      ipv4,ipv6      v4:routing,      Ethernet97.106,
\n      107      107:107      ipv4,ipv6      v4:routing,      Ethernet97.107,
\n      108      108:108      ipv4,ipv6      v4:routing,      Ethernet97.108,
\n      109      109:109      ipv4,ipv6      v4:routing,      Ethernet97.109,
\n      110      110:110      ipv4,ipv6      v4:routing,      Ethernet97.110,
\n      111      111:111      ipv4,ipv6      v4:routing,      Ethernet97.111,
\n      112      112:112      ipv4,ipv6      v4:routing,      Ethernet97.112,
\n      113      113:113      ipv4,ipv6      v4:routing,      Ethernet97.113,
\n      114      114:114      ipv4,ipv6      v4:routing,      Ethernet97.114,
\n      115      115:115      ipv4,ipv6      v4:routing,      Ethernet97.115,
\n      mgmt      100:100      ipv4,ipv6      v4:no routing,      Management1
\n'
>>> show_vrf_clean = show_vrf[0]["output"]
>>>
>>> show_vrf_clean.splitlines()
[u'      101      101:101      ipv4,ipv6      v4:routing,
Ethernet97.101, ', u'      102      102:102      ipv4,ipv6      v4:routing,
Ethernet97.102, ', u'      103      103:103      ipv4,ipv6      v4:routing,
Ethernet97.103, ', u'      104      104:104      ipv4,ipv6      v4:routing,
Ethernet97.104, ', u'      105      105:105      ipv4,ipv6      v4:routing,
Ethernet97.105, ', u'      106      106:106      ipv4,ipv6      v4:routing,
Ethernet97.106, ', u'      107      107:107      ipv4,ipv6      v4:routing,
Ethernet97.107, ', u'      108      108:108      ipv4,ipv6      v4:routing,
Ethernet97.108, ', u'      109      109:109      ipv4,ipv6      v4:routing,
Ethernet97.109, ', u'      110      110:110      ipv4,ipv6      v4:routing,
Ethernet97.110, ', u'      111      111:111      ipv4,ipv6      v4:routing,
Ethernet97.111, ', u'      112      112:112      ipv4,ipv6      v4:routing,
Ethernet97.112, ', u'      113      113:113      ipv4,ipv6      v4:routing,
Ethernet97.113, ', u'      114      114:114      ipv4,ipv6      v4:routing,
Ethernet97.114, ', u'      115      115:115      ipv4,ipv6      v4:routing,
Ethernet97.115, ', u'      mgmt      100:100      ipv4,ipv6      v4:no routing,
Management1      ']
>>>
>>> type(show_vrf_clean.splitlines())
<type 'list'>
>>>
>>> for each_line in show_vrf_clean.splitlines():
    print each_line

      101      101:101      ipv4,ipv6      v4:routing,      Ethernet97.101,
      102      102:102      ipv4,ipv6      v4:routing,      Ethernet97.102,

```



```

103      103:103      ipv4,ipv6      v4:routing,      Ethernet97.103,
104      104:104      ipv4,ipv6      v4:routing,      Ethernet97.104,
105      105:105      ipv4,ipv6      v4:routing,      Ethernet97.105,
106      106:106      ipv4,ipv6      v4:routing,      Ethernet97.106,
107      107:107      ipv4,ipv6      v4:routing,      Ethernet97.107,
108      108:108      ipv4,ipv6      v4:routing,      Ethernet97.108,
109      109:109      ipv4,ipv6      v4:routing,      Ethernet97.109,
110      110:110      ipv4,ipv6      v4:routing,      Ethernet97.110,
111      111:111      ipv4,ipv6      v4:routing,      Ethernet97.111,
112      112:112      ipv4,ipv6      v4:routing,      Ethernet97.112,
113      113:113      ipv4,ipv6      v4:routing,      Ethernet97.113,
114      114:114      ipv4,ipv6      v4:routing,      Ethernet97.114,
115      115:115      ipv4,ipv6      v4:routing,      Ethernet97.115,
mgmt      100:100      ipv4,ipv6      v4:no routing,      Management1

>>> each_line
u'  mgmt      100:100      ipv4,ipv6      v4:no routing,      Management1  '
>>>
>>> each_line.split()
[u'mgmt', u'100:100', u'ipv4,ipv6', u'v4:no', u'routing,', u'Management1']

>>> each_line.split()[0]
u'mgmt'

>>> str(each_line.split()[0])
'mgmt'

>>> for each_line in show_vrf_clean.splitlines():
    print str(each_line.split()[0])

101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
mgmt

```

Now we have an idea on how to use jsonrpc to collect the show output in “text” format and parse the output using Python’s string processing modules.

Let’s update our script with a function for VRF scale.

```

import pprint
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

def vrf_scale(node):

```

```
show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
show_vrf_clean = show_vrf[0]["output"]

# Create a list to store the VRFs
vrfs = []
for line in show_vrf_clean.splitlines():
    fields = line.split()
    vrfs.append(fields[0])
return vrfs

node = Server("https://admin:admin@172.28.170.97/command-api")
vrfs = vrf_scale(node)

print ("Number of VRFs: %s" %(len(vrfs)))
print ("VRFs List")
pprint.pprint(vrfs)
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Number of VRFs: 16
VRFs List
[u'101',
 u'102',
 u'103',
 u'104',
 u'105',
 u'106',
 u'107',
 u'108',
 u'109',
 u'110',
 u'111',
 u'112',
 u'113',
 u'114',
 u'115',
 u'mgmt']
```

ARP Scale

In VRF environment, we have to calculate the number of ARP entries per VRF and add them to derive the total number of ARP entries.

Arista EOS Show Command

```
Arista-switch# show ip arp vrf 101 summary | json
{
  "dynamicEntries": 3,
  "ipV4Neighbors": [],
  "notLearnedEntries": 0,
  "totalEntries": 3,
  "staticEntries": 0
}
```

```

Arista-switch# show ip arp vrf 107 summary | json
{
    "dynamicEntries": 4,
    "ipV4Neighbors": [],
    "notLearnedEntries": 0,
    "totalEntries": 4,
    "staticEntries": 0
}

```

Develop Script

Since we already wrote a function called `vrf_scale` to identify the VRF names, we can use that function to get the list of VRFs and then we can write a program to identify the ARP scale using that list.

Create a new Python script from IDLE and save it as `arp_scale.py`.

```

import pprint
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

def vrf_scale(node):
    show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
    show_vrf_clean = show_vrf[0]["output"]
    vrfs = []
    for line in show_vrf_clean.splitlines():
        fields = line.split()
        vrfs.append(fields[0])
    return vrfs

def arp_scale(node, vrfs):
    for each_vrf in vrfs:
        show_arp = node.runCmds(1, ["show ip arp vrf " + each_vrf + " summary"])
        print show_arp[0]["dynamicEntries"]

# Define Connection Attributes
node = Server("https://admin:admin@172.28.170.97/command-api")

# Call VRF function and get the VRF names
vrfs = vrf_scale(node)

# Call ARP function
arp_scale(node, vrfs)

```

Save and run the script.

```

>>> ===== RESTART =====
>>>
2
2
2
2
2
2
2
2
2

```

```
2
2
2
2
2
2
2
2
3
```

Let us complete the script by adding these number of ARP entries per VRF and return only the total number of ARP entries from the function.

```
import pprint
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

def vrf_scale(node):
    show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
    show_vrf_clean = show_vrf[0]["output"]
    vrfs = []
    for line in show_vrf_clean.splitlines():
        fields = line.split()
        vrfs.append(fields[0])
    return vrfs

def arp_scale(node, vrfs):
    arp_count = 0
    for each_vrf in vrfs:
        show_arp = node.runCmds(1, ["show ip arp vrf " + each_vrf + " summary"])
        arp_count += show_arp[0]["dynamicEntries"]
    return arp_count

# Define Connection Attributes
node = Server("https://admin:admin@172.28.170.97/command-api")

# Call VRF function and get the VRF names
vrfs = vrf_scale(node)

# Call ARP function
arp_count = arp_scale(node, vrfs)

print ("Total Number of ARP entries %s" % (arp_count))
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Total Number of ARP entries 33
```

Routing Scale

In VRF environment, we have to calculate number of routes per VRF and add them to derive the total number of routes.

Arista EOS Show Command

```
Arista-switch# show ip route vrf 101 summary | json
```

```
{
  "maskLen": {
    "24": 626,
    "8": 2,
    "32": 207609,
    "31": 6
  },
  "totalRoutes": 208243,
  "staticNexthopGroup": 0,
  "bgpCounts": {
    "bgpExternal": 208229,
    "bgpInternal": 0,
    "bgpTotal": 208229
  },
}
```

```
Arista-switch# show ip route vrf 102 summary | json
```

```
{
  "maskLen": {
    "24": 626,
    "8": 2,
    "32": 9,
    "31": 6
  },
  "totalRoutes": 643,
  "staticNexthopGroup": 0,
  "bgpCounts": {
    "bgpExternal": 629,
    "bgpInternal": 0,
    "bgpTotal": 629
  },
}
```

Develop Script

We are going to use the same logic as `arp_scale.py` script to calculate the route scale. We use `vrf_scale` function to get the list of VRFs and then by using that list we will write a function for calculating route scale.

Create a new Python script from IDLE and save it as `route_scale.py`.

```
import pprint
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

def vrf_scale(node):
    show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
    show_vrf_clean = show_vrf[0]["output"]
    vrfs = []
    for line in show_vrf_clean.splitlines():
        fields = line.split()
        vrfs.append(fields[0])
    return vrfs

def route_scale(node, vrfs):
    route_count = 0
```

```
for each_vrf in vrfs:
    show_route = node.runCmds(1, ["show ip route vrf " + each_vrf + " summary"])
    route_count += show_route[0]["totalRoutes"]
return route_count

# Define Connection Attributes
node = Server("https://admin:admin@172.28.170.97/command-api")

# Call VRF function and get the VRF names
vrfs = vrf_scale(node)

# Call Route function
route_count = route_scale(node, vrfs)

print ("Total Number of IPv4 routes %s" % (route_count))
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Total Number of IPv4 routes 234
```

TCAM Scale

Arista EOS Show Command

```
Arista-switch# show platform trident tcam | json
% This is an unconverted command

Arista-switch# show platform trident tcam
=== TCAM summary for switch Linecard0/0 ===
TCAM group 20 uses 1 entry and can use up to 767 more.
  MLAG uses 1 entries.
TCAM group 10 uses 38 entries and can use up to 1754 more.
  Mlag control traffic uses 4 entries.
  CVX traffic uses 6 entries.
  L3 Control Priority uses 20 entries.
  IGMP Snooping Flooding uses 8 entries.
TCAM group 11 uses 58 entries and can use up to 1734 more.
  ACL Management uses 10 entries.
  L2 Control Priority uses 10 entries.
  Storm Control Management uses 2 entries.
  ARP Inspection uses 1 entries.
  L3 Routing uses 35 entries.
TCAM group 43 uses 1 entry and can use up to 767 more.
  Vxlan EFP uses 1 entries.
```

We will collect the “show platform” output in “text” format and parse through the string to collect the number of TCAM entries. We can pipe the show output to receive only the interesting lines.

```
Arista-switch# show platform trident tcam | include TCAM group
TCAM group 20 uses 1 entry and can use up to 767 more.
TCAM group 10 uses 38 entries and can use up to 1754 more.
TCAM group 11 uses 58 entries and can use up to 1734 more.
TCAM group 43 uses 1 entry and can use up to 767 more.
```

Develop Script

Create a new Python script from IDLE and save it as `tcam_scale.py`.

```
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

node = Server("https://admin:admin@172.28.170.98/command-api")

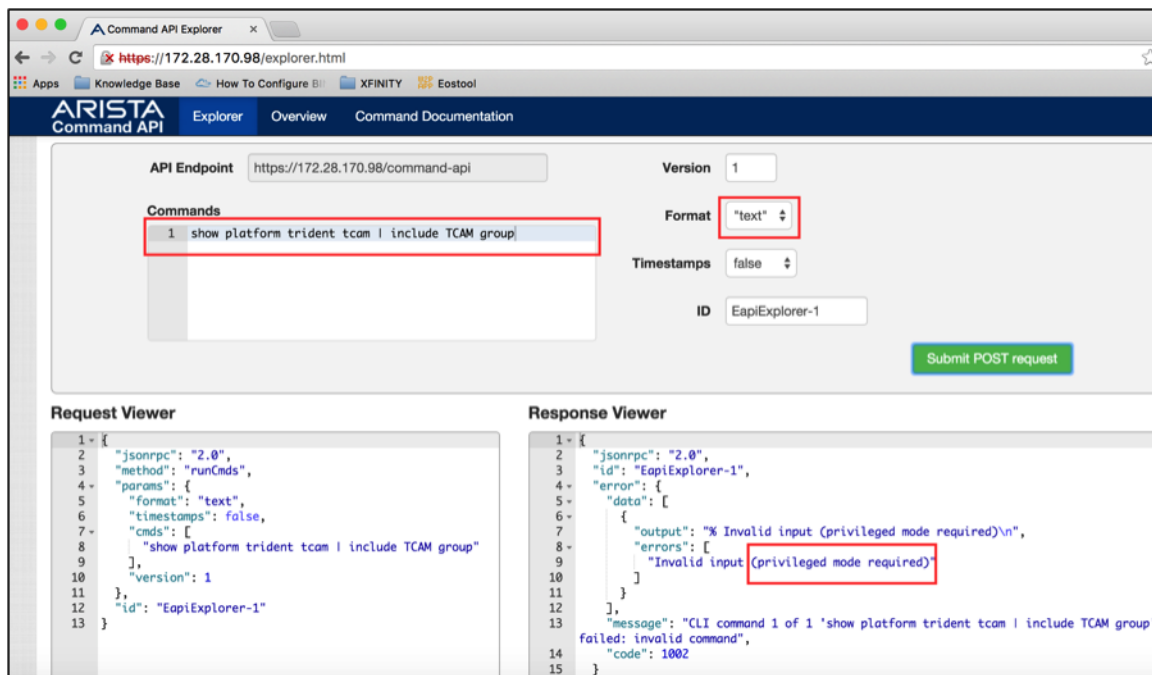
show_tcam = node.runCmds(1, ["show platform trident tcam | include TCAM group"], "text
→")
```

Save and run the script.

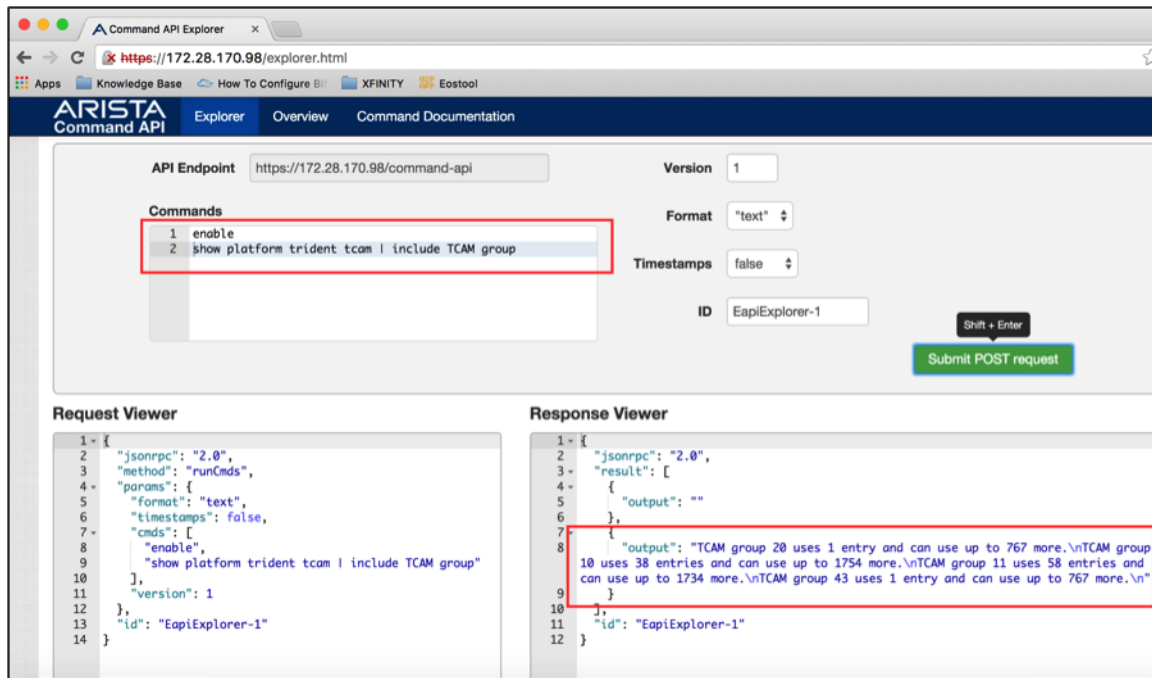
```
>> ===== RESTART =====
>>>

Traceback (most recent call last):
  File "/Users/anees/Google Drive/my-scripts/test.py", line 8, in <module>
    show_tcam = node.runCmds(1, ["show platform trident tcam | include TCAM group"],
    → "text")
ProtocolError: (1002, u"CLI command 1 of 1 'show platform trident tcam | include TCAM_
→group' failed: invalid command")
```

We need to understand why this command fails when we are running using jsonrpc. Let's open the command-api explorer for this switch and test the command.



The above output shows that the command should run from privileged mode. So we need to send the command `"enable"` in front of `"show platform trident tcam"` command.



Let's update the `tcam_scale.py` script to run the "show platform trident tcam" from privileged mode. Then we will explore the options to strip out the interesting data.

```
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

node = Server("https://admin:admin@172.28.170.98/command-api")

show_tcam = node.runCmds(1, ["enable", "show platform trident tcam | include TCAM_
↪group"], "text")
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
>>> show_tcam
[{'output': u''}, {'output': u'TCAM group 20 uses 1 entry and can use up to 767_
↪more.\nTCAM group 10 uses 38 entries and can use up to 1754 more.\nTCAM group 11_
↪uses 58 entries and can use up to 1734 more.\nTCAM group 43 uses 1 entry and can_
↪use up to 767 more.\n'}]
>>>
>>> show_tcam[1]
{'output': u'TCAM group 20 uses 1 entry and can use up to 767 more.\nTCAM group 10_
↪uses 38 entries and can use up to 1754 more.\nTCAM group 11 uses 58 entries and can_
↪use up to 1734 more.\nTCAM group 43 uses 1 entry and can use up to 767 more.\n'}
>>>
>>> show_tcam[1]["output"]
u'TCAM group 20 uses 1 entry and can use up to 767 more.\nTCAM group 10 uses 38_
↪entries and can use up to 1754 more.\nTCAM group 11 uses 58 entries and can use up_
↪to 1734 more.\nTCAM group 43 uses 1 entry and can use up to 767 more.\n'
>>>
>>> show_tcam[1]["output"].splitlines()
[u'TCAM group 20 uses 1 entry and can use up to 767 more.', u'TCAM group 10 uses 38_
↪entries and can use up to 1754 more.', u'TCAM group 11 uses 58 entries and can use_
↪up to 1734 more.', u'TCAM group 43 uses 1 entry and can use up to 767 more.']
```



```

>>>
>>> for each_line in show_tcam[1]["output"].splitlines():
    print each_line

TCAM group 20 uses 1 entry and can use up to 767 more.
TCAM group 10 uses 38 entries and can use up to 1754 more.
TCAM group 11 uses 58 entries and can use up to 1734 more.
TCAM group 43 uses 1 entry and can use up to 767 more.
>>>
>>> each_line
u'TCAM group 43 uses 1 entry and can use up to 767 more.'
>>>
>>> each_line.split()
[u'TCAM', u'group', u'43', u'uses', u'1', u'entry', u'and', u'can', u'use', u'up', u
↳ 'to', u'767', u'more.']
>>>
>>> each_line.split()[4]
u'1'
>>> for each_line in show_tcam[1]["output"].splitlines():
    print each_line.split()[4]

1
38
58
1
>>> tcam_count = 0
>>>
>>> for each_line in show_tcam[1]["output"].splitlines():
    tcam_count += each_line.split()[4]

Traceback (most recent call last):
  File "<pyshell#49>", line 2, in <module>
    tcam_count += each_line.split()[4]
TypeError: unsupported operand type(s) for +=: 'int' and 'unicode'
>>>
>>> for each_line in show_tcam[1]["output"].splitlines():
    tcam_count += int(each_line.split()[4])

>>> tcam_count
98

```

Let's update the tcam_scale.py script with a Python function.

```

from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

def tcam_scale(node):
    tcam_count = 0
    show_tcam = node.runCmds(1, ["enable", "show platform trident tcam | include TCAM_
↳ group"], "text")

    for each_line in show_tcam[1]["output"].splitlines():
        tcam_count += int(each_line.split()[4])

    return tcam_count

# Define Connection Attributes

```

```
node = Server("https://admin:admin@172.28.170.97/command-api")

# Call tcam scale function
tcam_count = tcam_scale(node)

print ("Total Number of TCAM Entries used is %s" % (tcam_count))
```

Hardware Scale

We will consolidate all the five scalability use cases in one script. Since we used jsonrpc for most of the scalability use cases, we will convert the mac scalability function to use jsonrpc instead of pyeapi. We will also add a function to find the hostname of the switch for storing the hardware scale under the switch name.

Create a new Python script from IDLE and save it as hardware_scale.py.

```
import pprint
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

def mac_scale(node):
    show_mac = node.runCmds(1, ["show mac address-table count"])
    show_mac_clean = show_mac[0]["vlanCounts"]
    mac_count = 0
    for each_vlan in show_mac_clean.keys():
        mac_count += show_mac_clean[each_vlan]["dynamic"]
    return mac_count

def vrf_scale(node):
    show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
    show_vrf_clean = show_vrf[0]["output"]
    vrfs = []
    for line in show_vrf_clean.splitlines():
        fields = line.split()
        vrfs.append(fields[0])
    return vrfs

def arp_scale(node, vrfs):
    arp_count = 0
    for each_vrf in vrfs:
        show_arp = node.runCmds(1, ["show ip arp vrf " + each_vrf + " summary"])
        arp_count += show_arp[0]["dynamicEntries"]
    return arp_count

def route_scale(node, vrfs):
    route_count = 0
    for each_vrf in vrfs:
        show_route = node.runCmds(1, ["show ip route vrf " + each_vrf + " summary"])
        route_count += show_route[0]["totalRoutes"]
    return route_count

def tcam_scale(node):
    tcam_count = 0
    show_tcam = node.runCmds(1, ["enable", "show platform trident tcam | include TCAM_↵group"], "text")
    for each_line in show_tcam[1]["output"].splitlines():
```

```

        tcam_count += int(each_line.split()[4])
    return tcam_count

def hostname(node):
    host_name = node.runCmds(1, ["show hostname"])
    host_name_clean = str(host_name[0]["hostname"])
    return host_name_clean

# Define Connection Attributes for jsonrpc
node = Server("https://admin:admin@172.28.170.97/command-api")

# Call the hardware scale functions
name = hostname(node)
mac_count = mac_scale(node)
vrfs = vrf_scale(node)
arp_count = arp_scale(node, vrfs)
route_count = route_scale(node, vrfs)
tcam_count = tcam_scale(node)

# Store the values in a dictionary
verify_scale = {}
verify_scale[name] = {"MAC Scale": mac_count,
                      "Number of VRFs": len(vrfs),
                      "Number of ARP Entries": arp_count,
                      "Number of Routes": route_count,
                      "Number of TCAM entries used": tcam_count
                      }

# Print the result
pprint.pprint(verify_scale)

```

Save and the run the script.

```

>>> ===== RESTART =====
>>>
{'22sw2': {'MAC Scale': 5,
           'Number of ARP Entries': 33,
           'Number of Routes': 234,
           'Number of TCAM entries used': 98,
           'Number of VRFs': 16}}

```

Now we have the logic for the script to find the hardware scale. Next we will integrate this script with our framework. We will start with section 1, 2 and 3 of the script framework.

```

"""
Script for hardware scalability assessment
"""

### Section 1

import pprint
from jsonrpclib import Server
import getpass
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

### Section 2

```

```
# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []
with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

### Section 3

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")
```

Let's add all the scalability assessment functions in section 4A.

```
## Section 4A - Define Hardware Scalability Assessment Functions

def mac_scale(node):
    show_mac = node.runCmds(1, ["show mac address-table count"])
    show_mac_clean = show_mac[0]["vlanCounts"]
    mac_count = 0
    for each_vlan in show_mac_clean.keys():
        mac_count += show_mac_clean[each_vlan]["dynamic"]
    return mac_count

def vrf_scale(node):
    show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
    show_vrf_clean = show_vrf[0]["output"]
    vrfs = []
    for line in show_vrf_clean.splitlines():
        fields = line.split()
        vrfs.append(fields[0])
    return vrfs

def arp_scale(node, vrfs):
    arp_count = 0
    for each_vrf in vrfs:
        show_arp = node.runCmds(1, ["show ip arp vrf " + each_vrf + " summary"])
        arp_count += show_arp[0]["dynamicEntries"]
    return arp_count

def route_scale(node, vrfs):
    route_count = 0
    for each_vrf in vrfs:
        show_route = node.runCmds(1, ["show ip route vrf " + each_vrf + " summary"])
        route_count += show_route[0]["totalRoutes"]
    return route_count

def tcam_scale(node):
    tcam_count = 0
```

```

    show_tcam = node.runCmds(1, ["enable", "show platform trident tcam | include TCAM_
↪group"], "text")
    for each_line in show_tcam[1]["output"].splitlines():
        tcam_count += int(each_line.split()[4])
    return tcam_count

def hostname(node):
    host_name = node.runCmds(1, ["show hostname"])
    host_name_clean = str(host_name[0]["hostname"])
    return host_name_clean

```

Our script for hardware scalability assessment is written for a single switch and we hard coded switch IP, username and password in the script. We need to redefine the jsonrpc connection object using variables.

We are going to rewrite this statement

```
node = Server("https://admin:admin@172.28.170.97/command-api")
```

with variables as below.

```
node = Server("https://" + my_username + ":" + my_password + "@" + switch + "/command-api")
```

Since we need to run this main script for all the switches, we will create a for loop and place the main script that calls all the functions within the for loop.

```

### Section 4B - Main Script

verify_scale = {}

for switch in switches:

    # Define Connection Attributes for jsonrpc
    node = Server("https://" + my_username + ":" + my_password + "@" + switch + "/command-api")

    # Call the hardware scale functions
    name = hostname(node)
    mac_count = mac_scale(node)
    vrfs = vrf_scale(node)
    arp_count = arp_scale(node, vrfs)
    route_count = route_scale(node, vrfs)
    tcam_count = tcam_scale(node)

    # Store the values in a dictionary

    verify_scale[name] = {"MAC Scale": mac_count,
                          "Number of VRFs": len(vrfs),
                          "Number of ARP Entries": arp_count,
                          "Number of Routes": route_count,
                          "Number of TCAM entries used": tcam_count
                        }

```

Finally we will print the result in the section 5 using pprint module.

```

### Section 5

# Print the result
pprint.pprint(verify_scale)

```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Enter your username: admin
{'22sw2': {'MAC Scale': 5,
          'Number of ARP Entries': 33,
          'Number of Routes': 234,
          'Number of TCAM entries used': 98,
          'Number of VRFs': 16},
 '22sw35': {'MAC Scale': 0,
            'Number of ARP Entries': 2050,
            'Number of Routes': 6307,
            'Number of TCAM entries used': 510,
            'Number of VRFs': 17},
 '22sw37': {'MAC Scale': 0,
            'Number of ARP Entries': 2046,
            'Number of Routes': 6273,
            'Number of TCAM entries used': 510,
            'Number of VRFs': 18}}
```

Exception Handling

One final feature we need to add in our script is exception handling. Without the exception handling, the script will terminate even if one switch is not accessible or any one EOS command is incorrect. Our goal is to save the error in a variable and continue executing the program for other switches or EOS commands. Also our goal is to differentiate if it is a connectivity (or access) issue or incorrect EOS command.

To understand the syntax of output error for inaccessible switch and incorrect EOS commands, we will test the script for those errors. We create a small script called `jsonexception.py` with incorrect EOS command “show hostname1”.

```
import pprint
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

node = Server("https://admin:admin@172.28.170.97/command-api")
host_name = node.runCmds(1, ["show hostname1"])
```

Save and run the script.

```
>>> ===== RESTART =====
>>>

Traceback (most recent call last):
  File "/Users/anees/jsonexception.py", line 9, in <module>
    host_name = node.runCmds(1, ["show hostname1"])
ProtocolError: (1002, u"CLI command 1 of 1 'show hostname1' failed: invalid command")

>>> import jsonrpclib
>>> dir(jsonrpclib)
['Config', 'Fault', 'History', 'MultiCall', 'ProtocolError', 'Server', '__builtins__',
↪ '__doc__', '__file__', '__name__', '__package__', '__path__', 'config', 'dumps',
↪ 'history', 'jsonclass', 'jsonrpc', 'loads']
```

Now we will update the script to handle this exception message “ProtocolError”.

```
import pprint
from jsonrpclib import Server, ProtocolError
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

try:
    node = Server("https://admin:admin@172.28.170.97/command-api")
    host_name = node.runCmds(1, ["show hostname1"])

except ProtocolError as e:
    print ("Invalid EOS Command %s" % (e))
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
Invalid EOS Command (1002, u"CLI command 1 of 1 'show hostname1' failed: invalid_
↪command")
```

In this example, 172.28.170.98 switch is accessible but username and password are not admin/admin.

```
import pprint
from jsonrpclib import Server, ProtocolError

import ssl
ssl._create_default_https_context = ssl._create_unverified_context

try:
    node = Server("https://admin:admin@172.28.170.98/command-api")
    host_name = node.runCmds(1, ["show hostname"])

except ProtocolError as e:
    print ("Invalid EOS Command %s" % (e))
```

Save and run the script.

```
>>> ===== RESTART =====
>>>

Traceback (most recent call last):
  File "/Users/anees/jsonexception.py", line 7, in <module>
    host_name = node.runCmds(1, ["show hostname"])
  File "/Library/Python/2.7/site-packages/jsonrpclib/jsonrpc.py", line 288, in __call_
↪__
    return self.__send(self.__name, args)
  File "/Library/Python/2.7/site-packages/jsonrpclib/jsonrpc.py", line 237, in _
↪request
    response = self._run_request(request)
  File "/Library/Python/2.7/site-packages/jsonrpclib/jsonrpc.py", line 255, in _run_
↪request
    verbose=self.__verbose
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/
↪xmlrpclib.py", line 1280, in request
    return self.single_request(host, handler, request_body, verbose)
  File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/
↪xmlrpclib.py", line 1328, in single_request
    response.msg,
ProtocolError: <ProtocolError for admin:admin@172.28.170.98/command-api: 401_
↪Unauthorized>
```

As you can see, “except ProtocolError” does not catch this exception of incorrect authentication and the script fails. We will create a except clause to catch all the other error messages.

```
import pprint
from jsonrpclib import Server, ProtocolError

import ssl
ssl._create_default_https_context = ssl._create_unverified_context

try:
    node = Server("https://admin:admin@172.28.170.98/command-api")
    host_name = node.runCmds(1, ["show hostname"])

except ProtocolError as e:
    print ("Invalid EOS Command %s" % (e))

except:
    print ("eAPI Connection Error")
```

Save and run the script.

```
>>> ===== RESTART =====
>>>
eAPI Connection Error
```

Let’s update the section 1 & 4B of the script with the exception handling method.

```
### Section 1

import pprint
from jsonrpclib import Server, ProtocolError
import getpass
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

### Section 4B - Main Program

verify_scale = {}

for switch in switches:
    try:
        # Define Connection Attributes for jsonrpc
        node = Server("https://" + my_username + ":" + my_password + "@" + switch + "/command-api"
↪")

        # Call the hardware scale functions
        name = hostname(node)
        mac_count = mac_scale(node)
        vrfs = vrf_scale(node)
        arp_count = arp_scale(node, vrfs)
        route_count = route_scale(node, vrfs)
        tcam_count = tcam_scale(node)

        # Store the values in a dictionary

        verify_scale[name] = {"MAC Scale": mac_count,
                               "Number of VRFs": len(vrfs),
```



```

        "Number of ARP Entries": arp_count,
        "Number of Routes": route_count,
        "Number of TCAM entries used": tcam_count
    }

    except ProtocolError as e:
        verify_scale[switch] = "Invalid EOS Command" + str(e)

    except:
        verify_scale[switch] = "eAPI Connection Error"

```

Now let us test the script. On the switches.txt file, we are going to add an IP address “172.28.170.98” which we know that having an authentication issue.

```

172.28.170.97
172.28.170.98
172.28.170.115
172.28.170.114

```

Save and run the script.

```

>>> ===== RESTART =====
>>>
Enter your username: admin
{'172.28.170.98': 'eAPI Connection Error',
 '22sw2': {'MAC Scale': 5,
           'Number of ARP Entries': 33,
           'Number of Routes': 234,
           'Number of TCAM entries used': 98,
           'Number of VRFs': 16},
 '22sw35': {'MAC Scale': 0,
            'Number of ARP Entries': 2050,
            'Number of Routes': 6307,
            'Number of TCAM entries used': 510,
            'Number of VRFs': 17},
 '22sw37': {'MAC Scale': 0,
            'Number of ARP Entries': 2046,
            'Number of Routes': 6273,
            'Number of TCAM entries used': 510,
            'Number of VRFs': 18}}

```

Final Script:

```

"""
Script for hardware scalability assessment
"""

### Section 1

import pprint
from jsonrpclib import Server, ProtocolError
import getpass
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

### Section 2

```

```
# Define file path and file names

file_path = "/Users/anees/Google Drive/my-scripts/"
file_name_switches = "switches.txt"
file_switches = file_path + file_name_switches

# Read the content of the file and save it in a List

switches = []
with open(file_switches) as readfile:
    for line in readfile:
        switches.append(line.strip())

### Section 3

# Input Username and Password

my_username = raw_input("Enter your username: ")
my_password = getpass.getpass("Enter your password: ")

### Section 4A - Define Hardware Scale Functions

def mac_scale(node):
    show_mac = node.runCmds(1, ["show mac address-table count"])
    show_mac_clean = show_mac[0]["vlanCounts"]
    mac_count = 0
    for each_vlan in show_mac_clean.keys():
        mac_count += show_mac_clean[each_vlan]["dynamic"]
    return mac_count

def vrf_scale(node):
    show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
    show_vrf_clean = show_vrf[0]["output"]
    vrfs = []
    for line in show_vrf_clean.splitlines():
        fields = line.split()
        vrfs.append(fields[0])
    return vrfs

def arp_scale(node, vrfs):
    arp_count = 0
    for each_vrf in vrfs:
        show_arp = node.runCmds(1, ["show ip arp vrf " + each_vrf + " summary"])
        arp_count += show_arp[0]["dynamicEntries"]
    return arp_count

def route_scale(node, vrfs):
    route_count = 0
    for each_vrf in vrfs:
        show_route = node.runCmds(1, ["show ip route vrf " + each_vrf + " summary"])
        route_count += show_route[0]["totalRoutes"]
    return route_count

def tcam_scale(node):
    tcam_count = 0
    show_tcam = node.runCmds(1, ["enable", "show platform trident tcam | include TCAM_↵group"], "text")
```

```

    for each_line in show_tcam[1]["output"].splitlines():
        tcam_count += int(each_line.split()[4])
    return tcam_count

def hostname(node):
    host_name = node.runCmds(1, ["show hostname"])
    host_name_clean = str(host_name[0]["hostname"])
    return host_name_clean

### Section 4B - Main Program

verify_scale = {}

for switch in switches:
    try:
        # Define Connection Attributes for jsonrpc
        node = Server("https://" + my_username + ":" + my_password + "@" + switch + "/command-api
↪")

        # Call the hardware scale functions
        name = hostname(node)
        mac_count = mac_scale(node)
        vrfs = vrf_scale(node)
        arp_count = arp_scale(node, vrfs)
        route_count = route_scale(node, vrfs)
        tcam_count = tcam_scale(node)

        # Store the values in a dictionary
        verify_scale[name] = {"MAC Scale": mac_count,
                              "Number of VRFs": len(vrfs),
                              "Number of ARP Entries": arp_count,
                              "Number of Routes": route_count,
                              "Number of TCAM entries used": tcam_count
                              }

    except ProtocolError as e:
        verify_scale[switch] = "Invalid EOS Command" + str(e)

    except:
        verify_scale[switch] = "eAPI Connection Error"

### Section 5

# Print the result
pprint.pprint(verify_scale)

```

Chapter 5: Web2Py Introduction

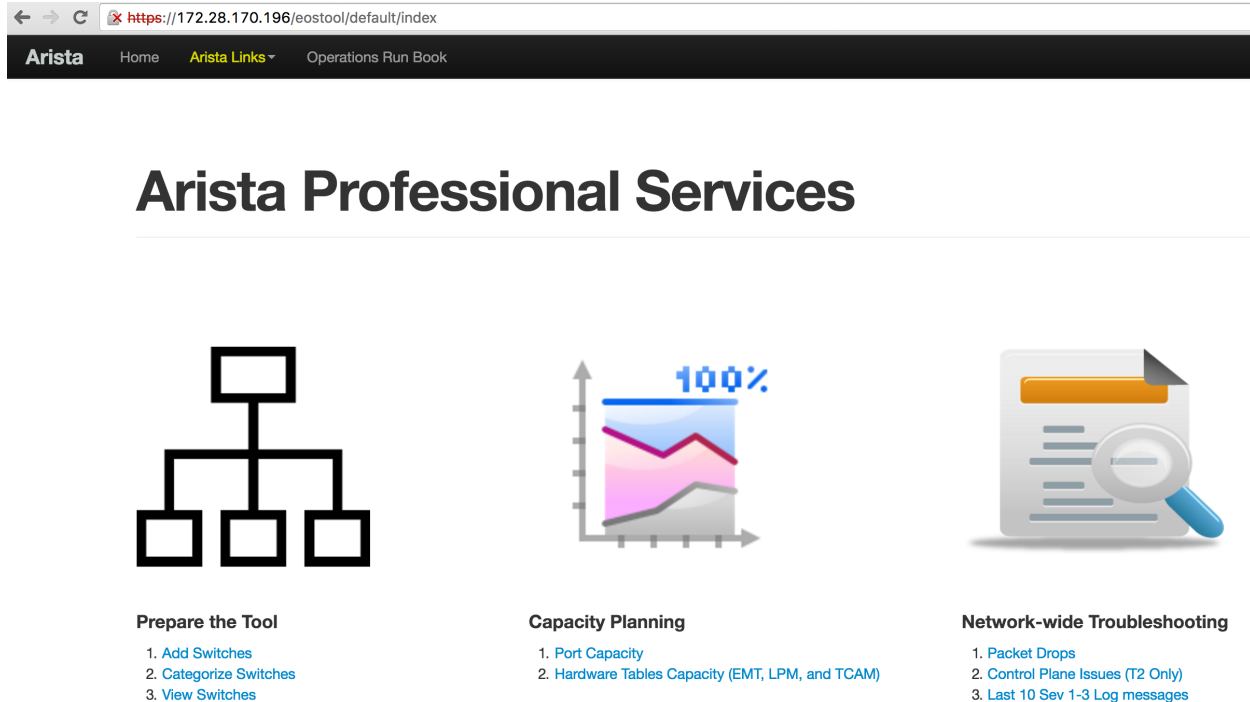
- *Understanding the functionality of the Tool*
 - *Prepare the Tool*
 - * *Add Switches*
 - * *Categorize Switches*
 - * *View Switches*
 - *Capacity Planning*
 - * *Port Capacity*
 - *Network-wide Troubleshooting*
 - * *Packet Drops*
- *Web2Py Framework for the Tool*
 - *Web2Py Framework*
 - *Controller*
 - *Views*

I hope that you are comfortable in writing Python scripts to automate some of the network operational tasks by now. So far we are writing the code and storing in a Python (.py) file and run the script from terminal or from any Python development environment. Next step is to improve the usability of the scripts you have written so far. How about if you can host all the Python scripts you have written so far in a web based tool? This way you can access the tool anywhere from the network using your browser. And you can share the tool with your team.

As you start learning Python, you will start to consume lot of Python modules written by the community. With the web based tool, you have to install those modules only on the servers where you are hosting your Python scripts. All the users who are consuming the tool does not have to install any of these modules and they don't even need to have Python installed on their systems.

What do we need to do to build a web based tool and host the Python scripts?. We need a web framework such as Web2Py or Django. In this book, we are going to use Web2Py as our web framework. Since the fundamental principal of this book is to teach you Python and web2py by using the networking use cases, we are going to explain the web2py framework by using the tool we are going to build.

We are going to build a tool like this:



We will first understand the tool and then we will review how it is implemented using web2py framework. Primary purpose of the tool is to host your Python scripts. As you can see that some of the scripts we created in the previous chapters were listed under network-wide troubleshooting and capacity planning sections. We are going to add a provision in the tool where you can manually add the IP addresses of the switches in the network. Prepare the tool section allows you to add IP addresses of the switches to the tool.

Once you added the list of switches in the tool, you can run your tasks under capacity planning and network-wide troubleshooting against those switches. We will walk through the tool to better understand what we are going to build using web2py framework.

Understanding the functionality of the Tool

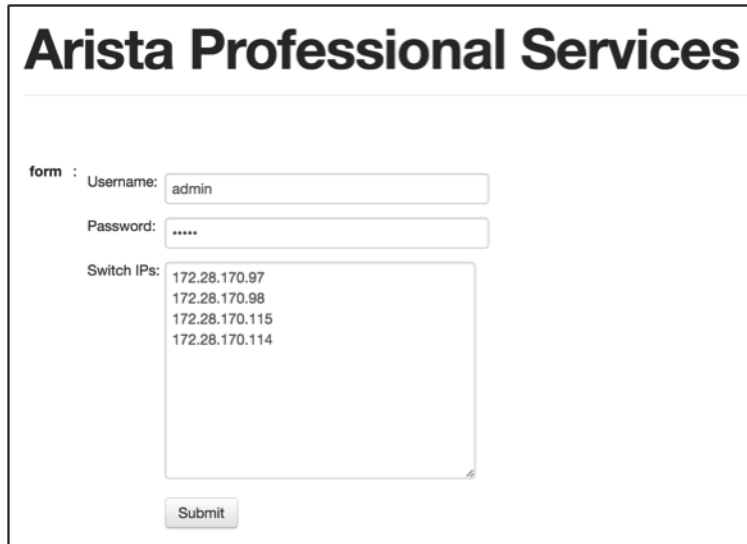
The purpose of the tool is to run common network operational tasks across multiple Arista switches in the network.

Prepare the Tool

Users should be able to add switches to the tool as and when they roll out new Arista switches in the network. We are also going to add an option to categorize switches based on site and role. This gives an option to run the operational tasks selectively on the switches belong to specific site and role.

Add Switches

Add switches will allow you to add the list of IP addresses of the Arista switches. It requires username and password as well.



The screenshot shows a web form titled "Arista Professional Services". The form is labeled "form :" and contains three input fields: "Username:" with the value "admin", "Password:" with the value "*****", and "Switch IPs:" with a list of four IP addresses: "172.28.170.97", "172.28.170.98", "172.28.170.115", and "172.28.170.114". A "Submit" button is located at the bottom of the form.

After you provide all information and hit submit, the script will collect switch name, model and software version from the switches and then it stores the data in a file locally in the server. We will show you the file name and the path when we show you how to build this tool later in this book. The add switches task also displays the inventory of the switches we are adding in the output as below.

```
error      :
inventory  : 172.28.170.114 :
               Role      : spine
               Site      : none
               hostname   : 22sw35
               model      : DCS-7250QX-64
               version    : 4.15.3F

               172.28.170.115 :
               Role      : spine
               Site      : none
               hostname   : 22sw37
               model      : DCS-7250QX-64
               version    : 4.15.3F

               172.28.170.97  :
               Role      : spine
               Site      : none
               hostname   : 22sw2
               model      : DCS-7050SX-128
               version    : 4.15.3F

               172.28.170.98  :
               Role      : spine
               Site      : none
               hostname   : 22sw4
               model      : DCS-7050SX-128
               version    : 4.15.3F
```

Categorize Switches

When you click “Categorize Switches”, the script will pull the inventory information of the switches we added in the “Add Switches” task from the file stored in the server.

form :

22sw4 Site:	<input type="text" value="sjc"/>
22sw4 Role:	<input type="text" value="leaf"/>
22sw37 Site:	<input type="text" value="sjc"/>
22sw37 Role:	<input type="text" value="spine"/>
22sw35 Site:	<input type="text" value="sjc"/>
22sw35 Role:	<input type="text" value="spine"/>
22sw2 Site:	<input type="text" value="sjc"/>
22sw2 Role:	<input type="text" value="leaf"/>

For each switch, it will show the configured site and role. For the switches added through “Add Switches” task, these two fields will be configured as none. You can manually update each switch with corresponding site and role as per your network design.

View Switches

You can view the current inventory of the switches using “View Switches” task.

```
inventory : 172.28.170.114 :  
              Role      : spine  
              Site      : sjc  
              hostname   : 22sw35  
              model      : DCS-7250QX-64  
              version    : 4.15.3F  
  
            172.28.170.115 :  
              Role      : spine  
              Site      : sjc  
              hostname   : 22sw37  
              model      : DCS-7250QX-64  
              version    : 4.15.3F  
  
            172.28.170.97  :  
              Role      : leaf  
              Site      : sjc  
              hostname   : 22sw2  
              model      : DCS-7050SX-128  
              version    : 4.15.3F  
  
            172.28.170.98  :  
              Role      : leaf  
              Site      : sjc  
              hostname   : 22sw4  
              model      : DCS-7050SX-128  
              version    : 4.15.3F
```

Capacity Planning

We will just see how to run one of the tasks in this category to understand how the tool works.

Port Capacity

When you click “Port Capacity” task, it will prompt you for username, password, site and role. If you want to run the task on all the switches, select All for both site and role fields. Basically site and role fields give you the flexibility to select the list of switches to which you want to run the task.

form :

Username:

Password:

Network:

Role:

After you enter username, password, site and role, the script will run against the switches based on your selection and displays the inventory of unused ports.

```
unused_ports :
  22sw2 :
    10 :
      10GBASE-CR : 1
      10GBASE-SR : 1
      Not Present : 90
      dot1q-encapsulation : 3
    40 :
      Not Present : 6
    description : 96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU
    model : DCS-7050SX-128
    serialnumber : JPE14080457
  22sw4 :
    10 :
      10GBASE-CR : 1
      10GBASE-SR : 1
      Not Present : 90
      dot1q-encapsulation : 3
    40 :
      Not Present : 6
    description : 96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU
    model : DCS-7050SX-128
    serialnumber : JPE14080459
```

Network-wide Troubleshooting

We will just see how to run one of the tasks in this category to understand how the tool works.

Packet Drops

Almost all the use cases will prompt you the same form. The data you need to provide is the username and password of the switches and then select the site and role.

form : Username:

Password:

Network:

Role:

After you enter username, password, site and role, the script will run against the switches based on your selection. It displays the switch name and the interfaces where the packet drops are observed in the network.

```
interface_drops : 172.28.170.97 : Problem with eAPI connectivity or show command syntax error or Incorrect Authentication
                  22sw2       :
                  22sw35      :
                  22sw37      : Ethernet1/1 :
                                Interface Status : connected
                                Line Protocol   : up
                                Status           :
                                outDiscards      :
                                                OutputErrorsDetail : {u'lateCollisions': 0, u'deferredTransmissions': 0, u'bxPause': 0,
                                                u'collisions': 0}
                                                Total Discards      : 970

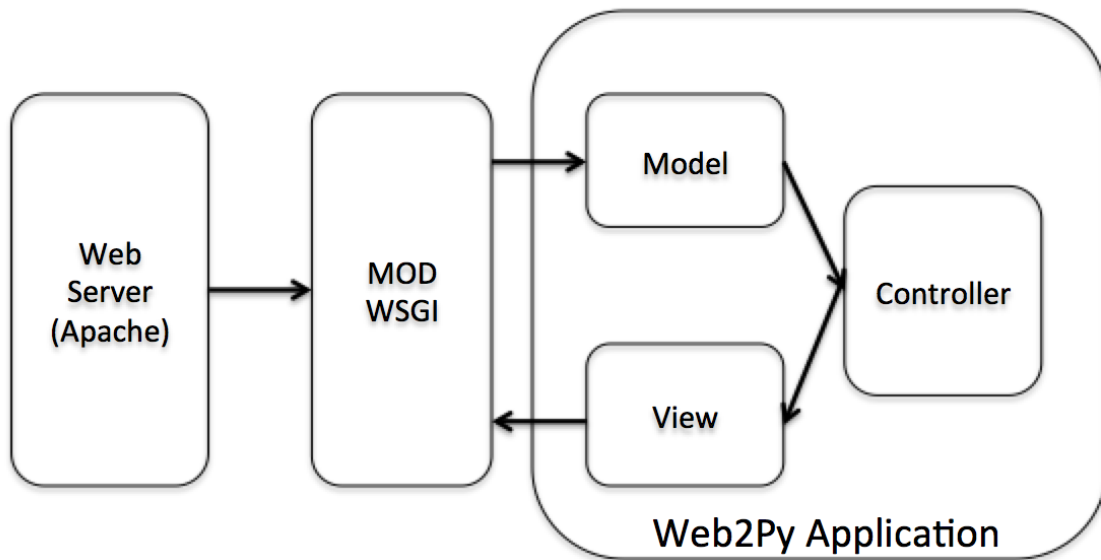
                  22sw4       : Ethernet97 :
                                Interface Status : connected
                                Line Protocol   : up
                                Status           :
                                outDiscards      :
                                                OutputErrorsDetail : {u'lateCollisions': 0, u'deferredTransmissions': 0, u'bxPause': 0,
                                                u'collisions': 0}
                                                Total Discards      : 32
```

Web2Py Framework for the Tool

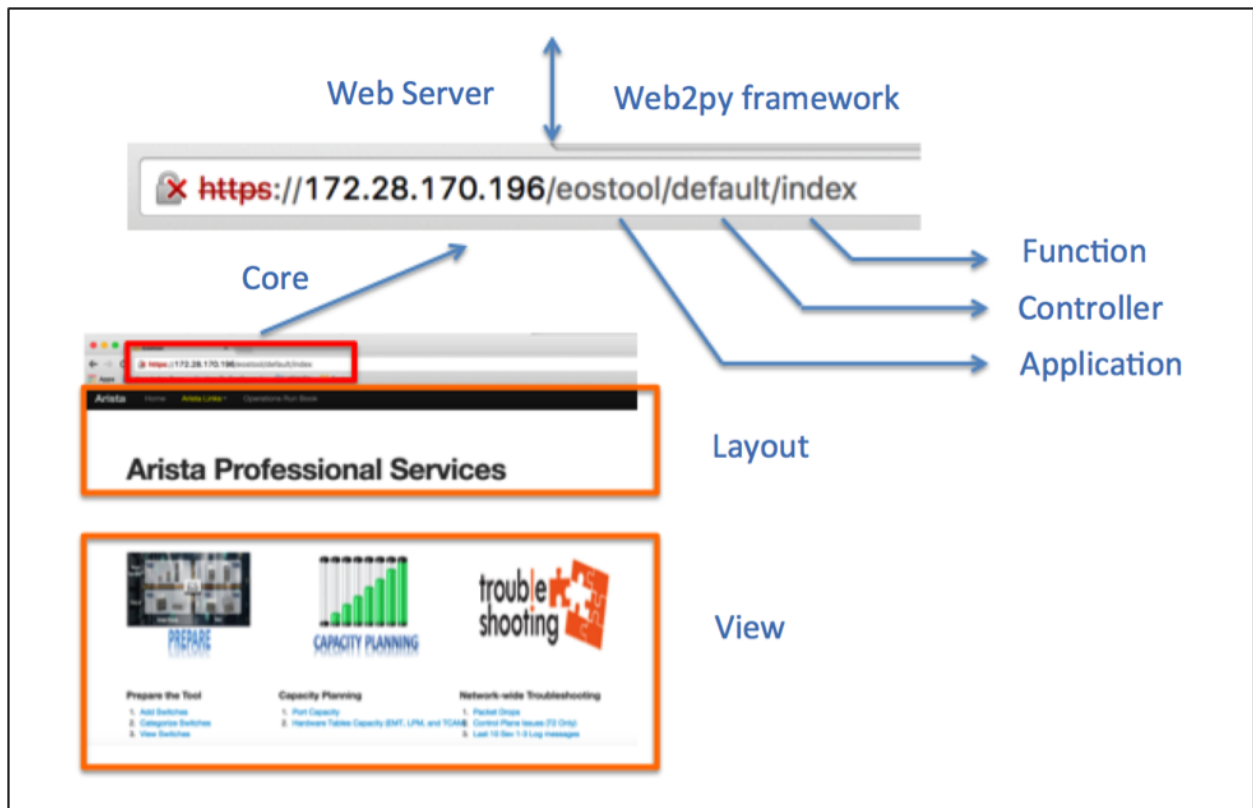
Web2Py can be downloaded from www.web2py.com and installed on most of the desktop and server operating systems. Web2Py can be considered as server side application systems and you need a web tier to serve the applications to clients. You can either use the Rocket WSGI web server that installed with Web2Py or you can leverage other web servers such as Apache. In our example, we will use Apache web server running on top of Ubuntu Linux operating systems to host our tool.

Web2Py Framework

The web2py instance can server multiple applications. Each application has a controller (default.py) where you write your Python scripts and a view which generates html page to interact with end users.

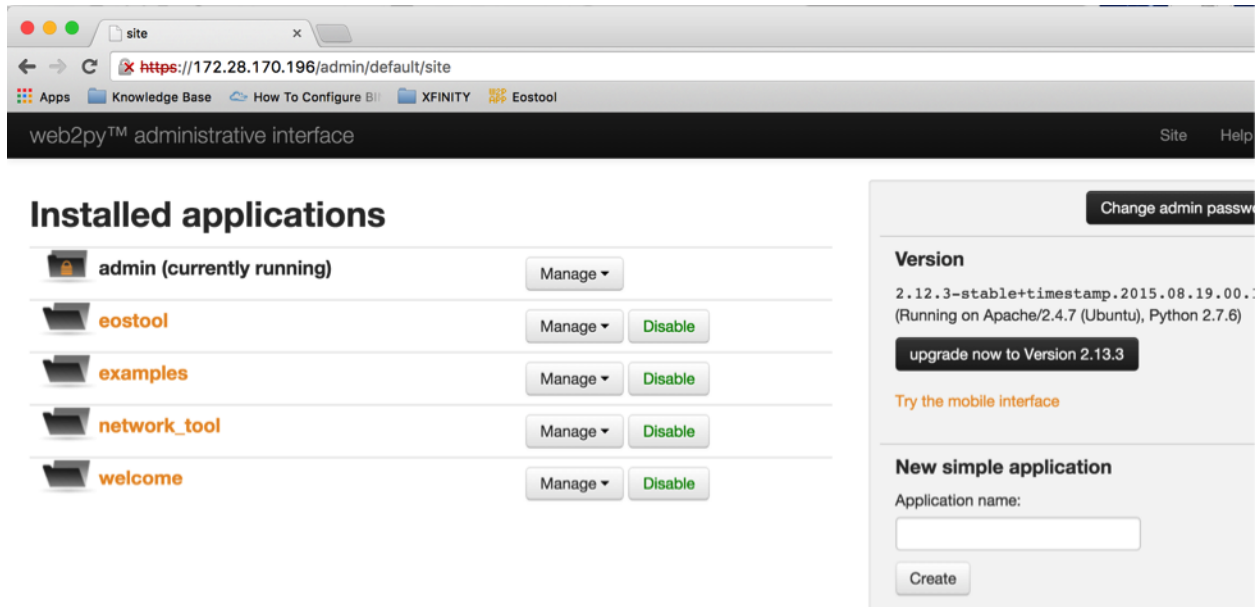


We are going to explore the web2py components of our specific application in this section. Then we will show you how to build the entire application from scratch in this book. Let us explore the web2py components from the front page of our tool.

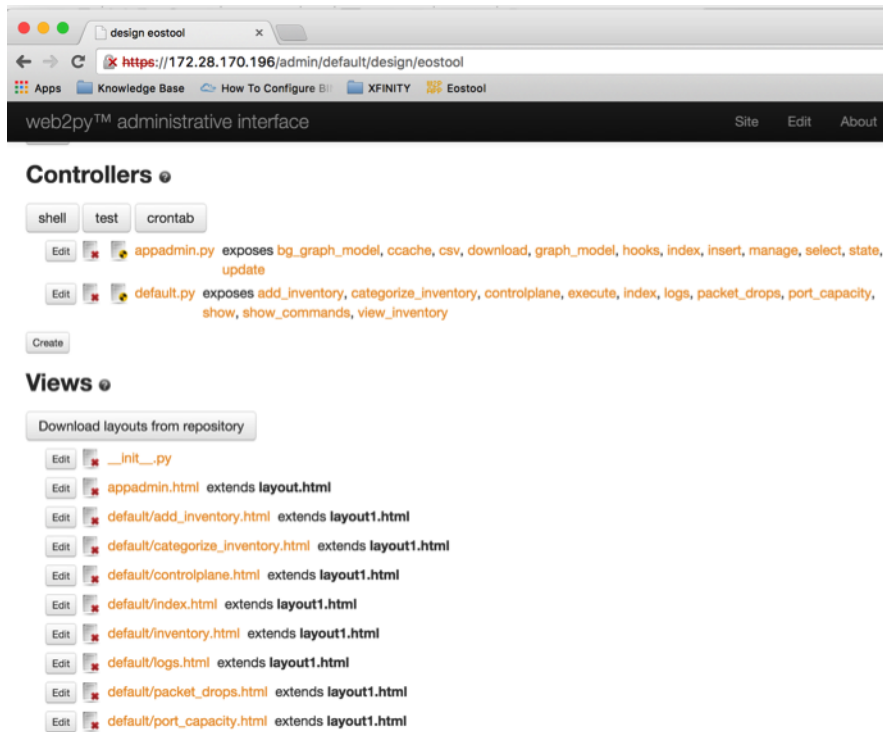


As you can see from the URL, our application name is “eostool”, controller name is “default” and the function name is “index”.

Once web2py is installed, you can create multiple applications. In our example, we created an application called “eostool”. The below screenshot shows the list of applications created in our web2py instance.

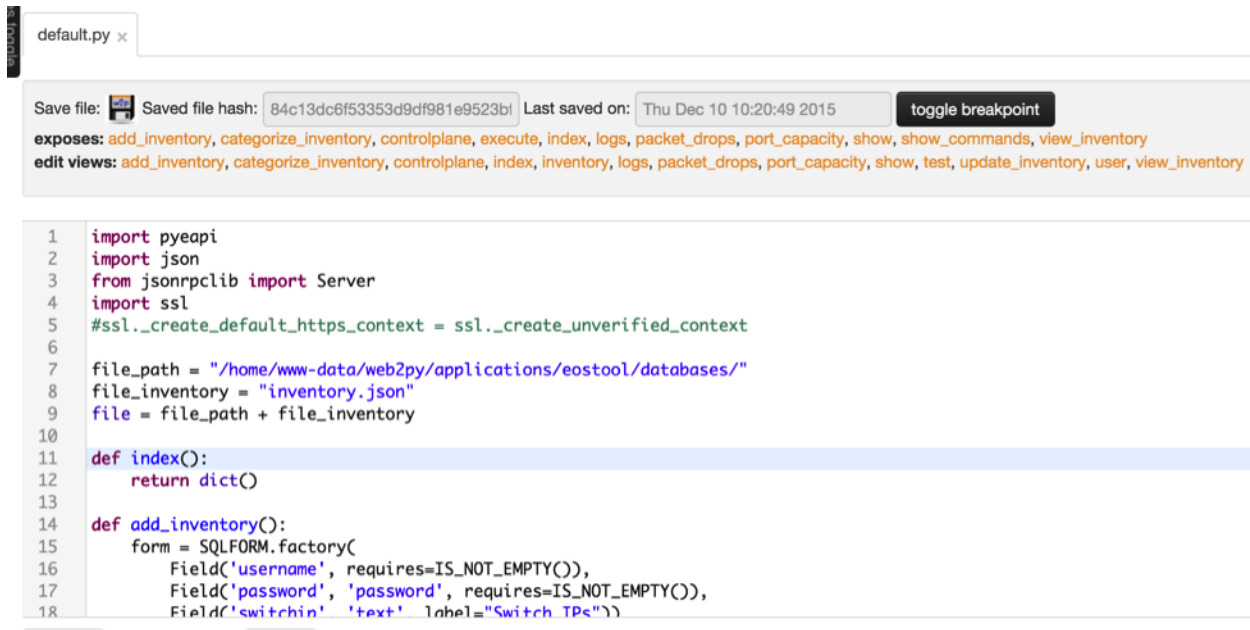


Using Web2Py’s web administrative interface, you can create applications. Each application follows MVC (Model, View and Controller) framework.



Controller

Default controller name for any web2py application is default.py. This is where we write all our Python scripts. You can edit default.py using web2py administrative interface.



```

1 import pyeapi
2 import json
3 from jsonrpclib import Server
4 import ssl
5 #ssl._create_default_https_context = ssl._create_unverified_context
6
7 file_path = "/home/www-data/web2py/applications/eostool/databases/"
8 file_inventory = "inventory.json"
9 file = file_path + file_inventory
10
11 def index():
12     return dict()
13
14 def add_inventory():
15     form = SQLFORM.factory(
16         Field('username', requires=IS_NOT_EMPTY()),
17         Field('password', 'password', requires=IS_NOT_EMPTY()),
18         Field('switchin', 'text', label="Switch IPs")

```

In the previous chapters, we created separate files for each of our use cases. In web2py, each of the use cases corresponds to a function in the default controller. Port capacity, hardware scalability assessment, data plane and control plane drops are all separate functions in the default controller.

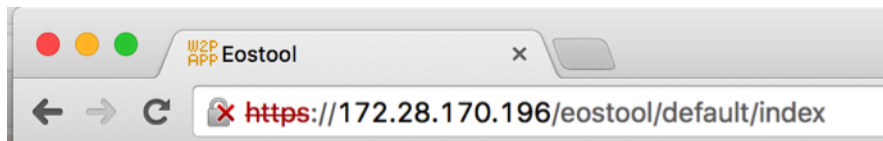
For example, Add Switches in our tool is a function called `add_inventory()` which is in the controller `default.py`.

Prepare the Tool

1. [Add Switches](#)
2. [Categorize Switches](#)
3. [View Switches](#)

https://172.28.170.196/eostool/default/add_inventory

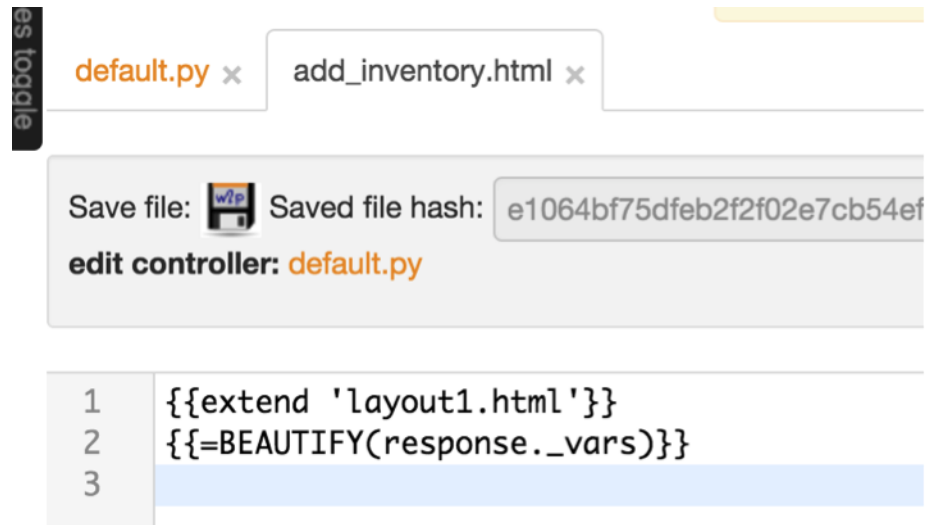
Home page of the application “eostool” is also a function called `index()` in the controller `default.py`.



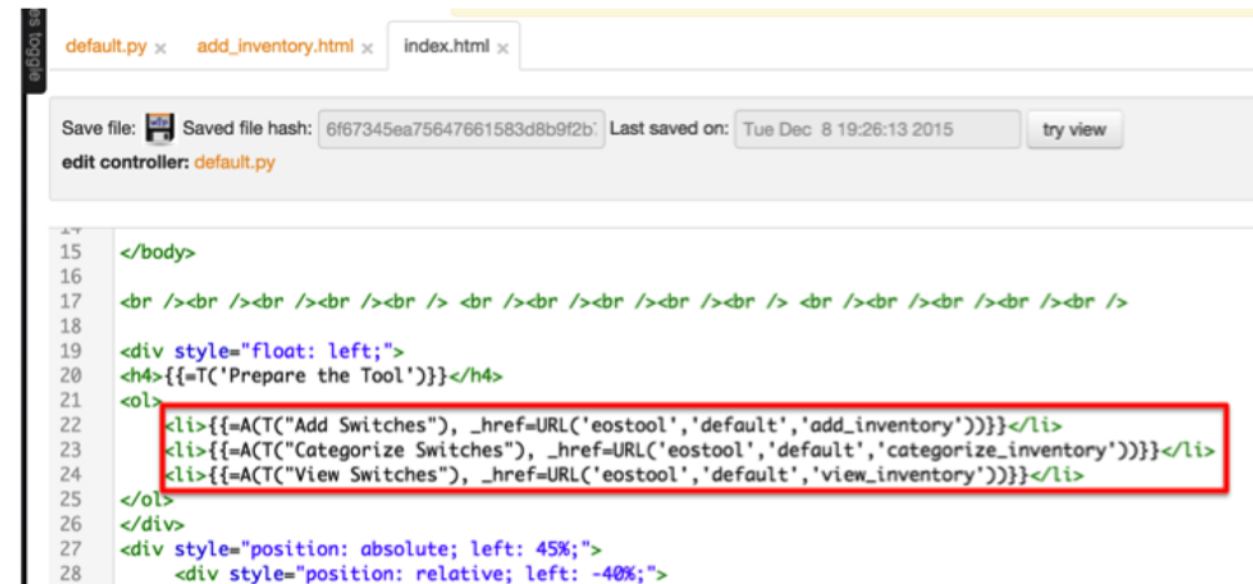
Views

The tool has to interact with the end users to collect the input and also to display the result of your Python script. As we know that each function in the default controller is your Python script performs specific network operations task. Each task requires user interface to collect the input from users and to display result. We will create “web2py view” for each function to facilitate the user interface. View is nothing but a html file and the name of the html file

is same as the name of the function in the default controller. For example view of the function `add_inventory()` is `add_inventory.html`.



Whenever you need to create a new network operations task (Python script), you will create a separate function in this `default.py` controller and a separate view for your function. Then you can add a link in the home page (`index.html`).



Chapter 6: Web2Py Installation

- *Install Required Packages*
 - *About My Linux*
 - *Install Required Packages for Python Development*
 - *Install Python modules*
 - *Install Apache*
 - *Create SSL Certificate*
- *Install Web2Py*
 - *Configure Apache to use mod_wsgi*
- *Start Web2Py Service*
 - *Verify Web2Py Portal*

Web2py is an open source full stack framework. You can download web2py from <http://web2py.com/init/default/download>. You can install it on Windows, Apple Mac or any of the Linux distributions. Installation instruction is documented here <http://web2py.com/books/default/chapter/29/13/deployment-recipes>. In this chapter, we will show you how to install web2py in Ubuntu. Below is the quick installation steps to install and setup web2py on Ubuntu running Apache as the web server.

Install Required Packages

About My Linux

```
anees@ubuntu-web2py:~$ uname -a
Linux ubuntu-web2py 3.16.0-30-generic #40~14.04.1-Ubuntu SMP Thu Jan 15 17:43:14 UTC
↪2015 x86_64 x86_64 x86_64 GNU/Linux
```

```
anees@ubuntu-web2py:~$ python --version
Python 2.7.6

anees@ubuntu-web2py:/$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:98:c8:32
          inet addr:172.28.170.197  Bcast:172.28.171.255  Mask:255.255.254.0
          inet6 addr: fe80::20c:29ff:fe98:c832/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:97480 errors:0 dropped:0 overruns:0 frame:0
          TX packets:58948 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:111218578 (111.2 MB)  TX bytes:10125393 (10.1 MB)
```

Install Required Packages for Python Development

```
anees@ubuntu-anees-1:~$ sudo apt-get install build-essential python-dev libsqlite3-
↳dev libreadline6-dev libgdbm-dev zlib1g-dev libbz2-dev sqlite3 zip libssl-dev
```

Install Python modules

```
anees@ubuntu-web2py:~$ sudo apt-get install python-pip

anees@ubuntu-web2py:~$ sudo pip install pyeapi

anees@ubuntu-web2py:~$ sudo pip install jsonrpc
```

Install Apache

```
anees@ubuntu-web2py:~$ sudo apt-get install apache2
anees@ubuntu-web2py:~$ sudo apt-get install libapache2-mod-wsgi
anees@ubuntu-web2py:~$ sudo a2enmod wsgi
anees@ubuntu-web2py:~$ sudo a2enmod ssl
anees@ubuntu-web2py:~$ sudo a2enmod proxy
anees@ubuntu-web2py:~$ sudo a2enmod proxy_http
anees@ubuntu-web2py:~$ sudo a2enmod headers
anees@ubuntu-web2py:~$ sudo a2enmod expires
anees@ubuntu-web2py:~$ sudo a2enmod rewrite
```

Create SSL Certificate

```
anees@ubuntu-web2py:~$ sudo mkdir /etc/apache2/ssl
anees@ubuntu-web2py:~$ sudo sh -c 'openssl genrsa 1024 > /etc/apache2/ssl/self_signed.
↳key'

anees@ubuntu-web2py:~$ sudo chmod 400 /etc/apache2/ssl/self_signed.key

anees@ubuntu-web2py:~$ sudo sh -c 'openssl req -new -x509 -nodes -sha1 -days 365 -key_
↳/etc/apache2/ssl/self_signed.key > /etc/apache2/ssl/self_signed.cert'
You are about to be asked to enter information that will be incorporated
```

```

into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:San Jose
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Arista
Organizational Unit Name (eg, section) []:Services
Common Name (e.g. server FQDN or YOUR name) []:ubuntu-web2py.mylab.com
Email Address []:admin@mylab.com

anees@ubuntu-web2py:~$ sudo sh -c 'sudo openssl x509 -noout -fingerprint -text < /etc/
↪apache2/ssl/self_signed.cert > /etc/apache2/ssl/self_signed.info'

```

Install Web2Py

```

anees@ubuntu-web2py:~$ cd /home
anees@ubuntu-web2py:/home$
anees@ubuntu-web2py:/home$ sudo mkdir www-data
anees@ubuntu-web2py:/home$ cd www-data/

anees@ubuntu-web2py:/home/www-data$ sudo wget http://web2py.com/examples/static/
↪web2py_src.zip

anees@ubuntu-web2py:/home/www-data$ sudo unzip web2py_src.zip
anees@ubuntu-web2py:/home/www-data$ sudo mv web2py/handlers/wsgihandler.py web2py/
↪wsgihandler.py

anees@ubuntu-web2py:/home/www-data$ sudo chown -R www-data:www-data web2py

```

Configure Apache to use mod_wsgi

```

anees@ubuntu-anees-1:~$ cd /etc/apache2/sites-available

anees@ubuntu-anees-1:/etc/apache2/sites-available$ sudo vi web2py.conf

WSGIDaemonProcess web2py user=www-data group=www-data processes=1 threads=1

<VirtualHost *:80>

    RewriteEngine On
    RewriteCond %{HTTPS} !=on
    RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]

    CustomLog /var/log/apache2/access.log common
    ErrorLog /var/log/apache2/error.log
</VirtualHost>

<VirtualHost *:443>
    SSLEngine on

```

```
SSLCertificateFile /etc/apache2/ssl/self_signed.cert
SSLCertificateKeyFile /etc/apache2/ssl/self_signed.key

WSGIProcessGroup web2py
WSGIScriptAlias / /home/www-data/web2py/wsgihandler.py
WSGIPassAuthorization On

<Directory /home/www-data/web2py>
    AllowOverride None
    Require all denied
    <Files wsgihandler.py>
        Require all granted
    </Files>
</Directory>

AliasMatch ^/([^/]+)/static/(?!_[\d]+\.[\d]+\.[\d]+)/?(.*) \
    /home/www-data/web2py/applications/$1/static/$2

<Directory /home/www-data/web2py/applications/*/static/>
    Options -Indexes
    ExpiresActive On
    ExpiresDefault "access plus 1 hour"
    Require all granted
</Directory>

CustomLog /var/log/apache2/ssl-access.log common
ErrorLog /var/log/apache2/error.log
</VirtualHost>

:wq!

anees@ubuntu-web2py:/etc/apache2/sites-available$ cd ..
anees@ubuntu-web2py:/etc/apache2$ cd sites-enabled/
anees@ubuntu-web2py:/etc/apache2/sites-enabled$ sudo rm *.*
anees@ubuntu-web2py:/etc/apache2/sites-enabled$ sudo a2ensite web2py

anees@ubuntu-anees-1:/etc/apache2/sites-available$ sudo service apache2 restart
```

Start Web2Py Service

```
anees@ubuntu-anees-1:/etc/apache2/sites-available$ cd /home/www-data/web2py

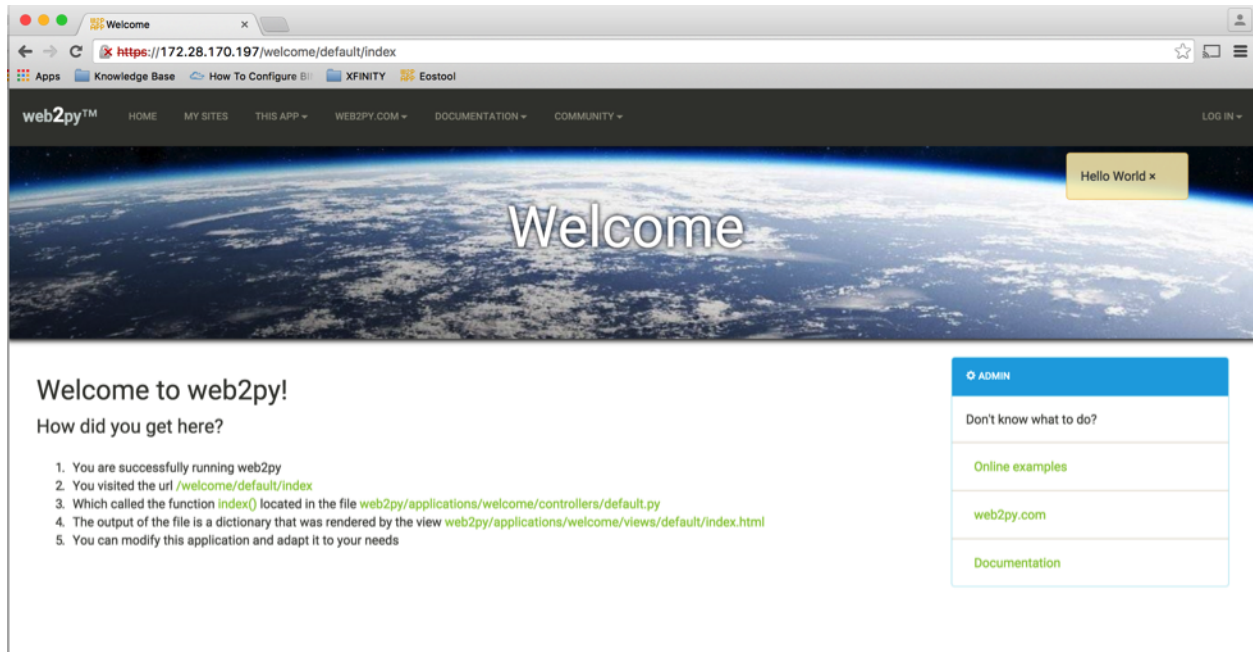
anees@ubuntu-anees-1:/home/www-data/web2py $
sudo -u www-data python -c "from gluon.widget import console; console();"

anees@ubuntu-anees-1:/home/www-data/web2py $
sudo -u www-data python -c "from gluon.main import save_password; save_password(raw_
↪input('admin password: '),443)"

**** You will be prompted to setup the web2py admin password ****
admin password:
```

Verify Web2Py Portal

Verify the web2py portal by launching from your browser. In our example, we will launch web2py portal using the url `https://172.28.170.197`.



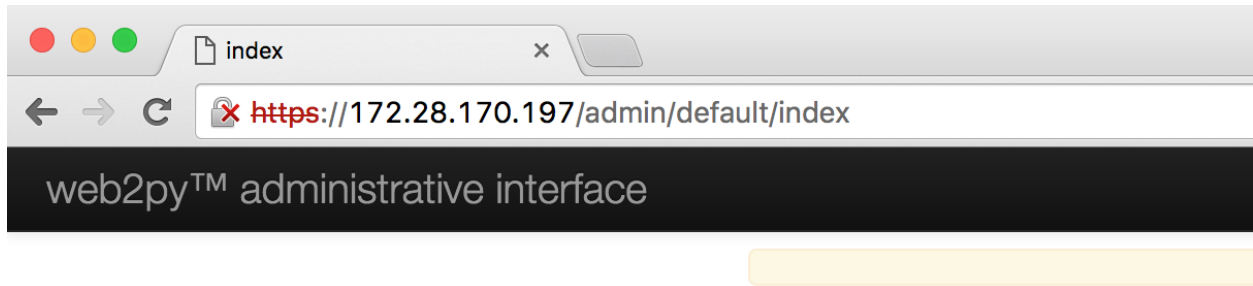
Chapter 7: Creating a New Web2Py Application

- *Default Admin Page*
- *Create a New Application*
- *Edit the Application*

We are going to create a new application and launch all our use cases through that application. You can create many applications and host it from single web2py instance. For example, you can build separate applications for your network operations, engineering and architecture groups. The application which we are going to create is more suitable for network operations team.

Default Admin Page

When you install Web2Py, it installs few applications by default. One of them is an admin application which provides the administrative interface to create, modify, develop and delete applications. You can access the admin application from your browser by using the url <https://<web-server>/admin/default/index>. It will prompt you for the admin password. Use the password you set when you brought the web2py service up.



web2py™ Web Framework

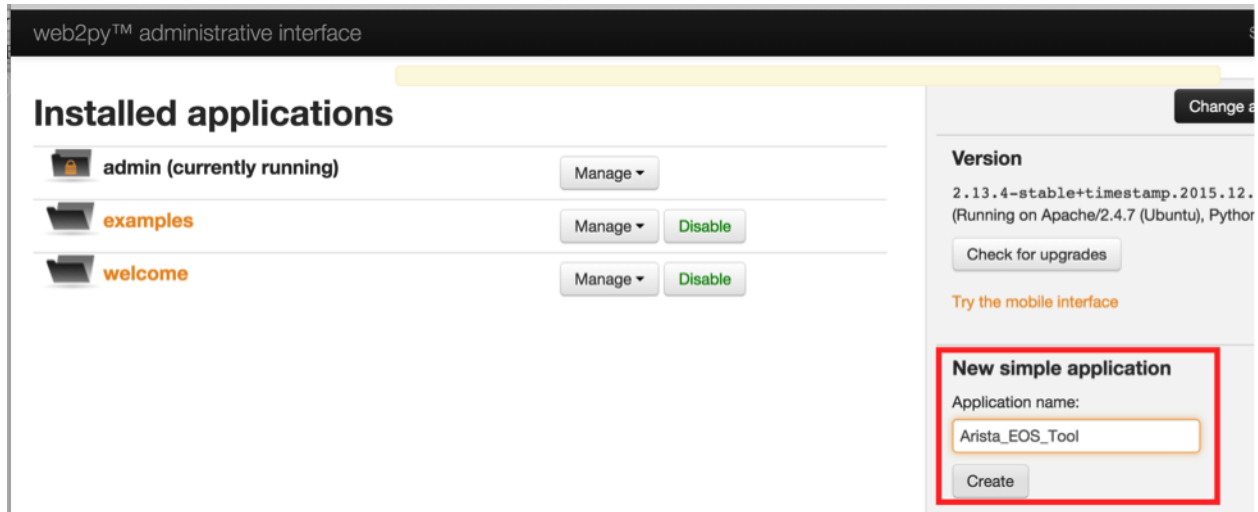
Login to the Administrative Interface

Administrator Password:

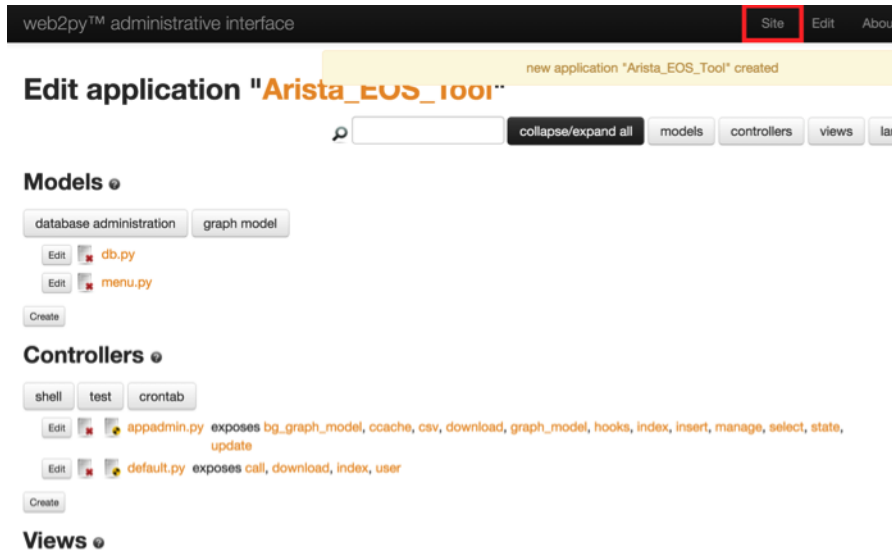
You will see the default applications created as part of the web2py installation.

Create a New Application

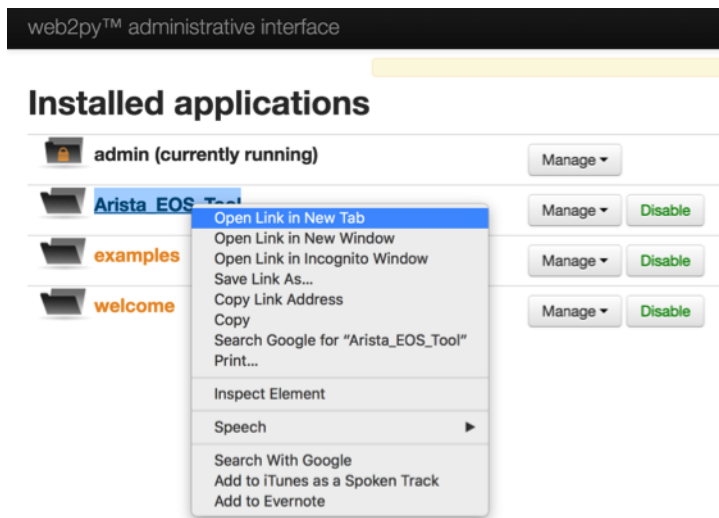
You can create new applications from the admin interface. Simply provide a name for your new application under “New simple application” section and click create. In our example, we use the application name as “Arista_EOS_Tool”.



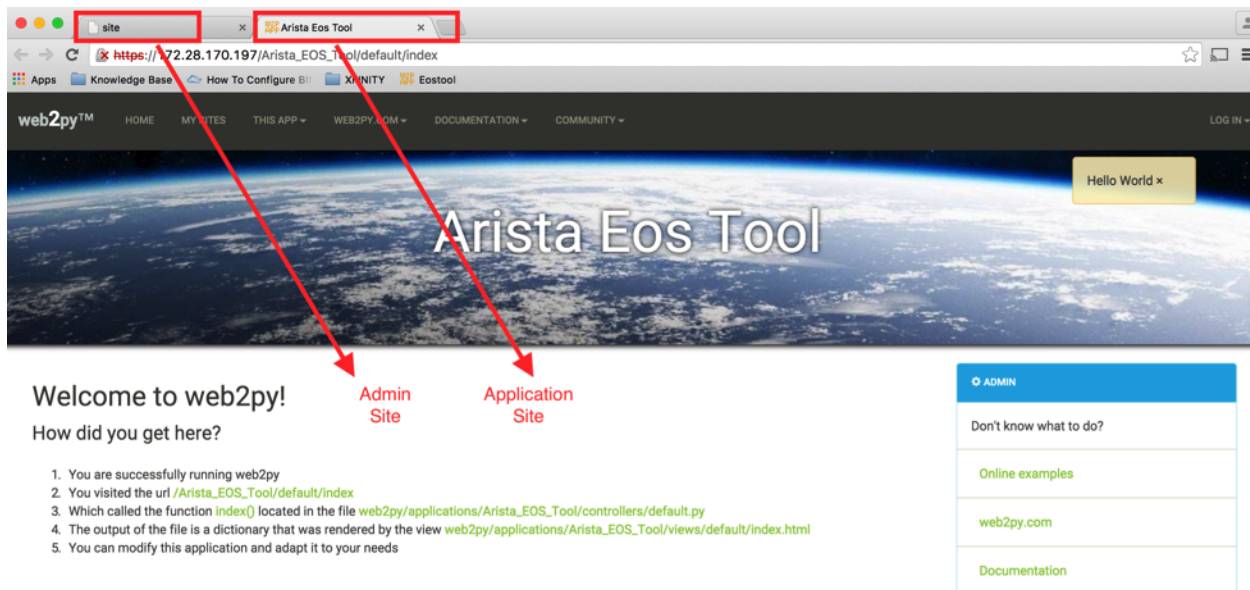
It will create the new application with default models, controllers and views.



You can view the newly created application by directly launching from your browser using the url https://<web-server>/Arista_EOS_Tool/default/index. You can also launch your application from admin interface. Right click your application and click “open Link in New Tab”.



This is the default web site created by web2py for your application.



You can verify the folders and files created by web2py for your new application in the Ubuntu server. You can see the default applications and your application in the “/home/www-data/web2py/applications” folder.

```
anees@ubuntu-web2py:/home/www-data/web2py/applications$ pwd
/home/www-data/web2py/applications

anees@ubuntu-web2py:/home/www-data/web2py/applications$ ll
drwxr-xr-x  8 www-data www-data 4096 Mar 14 16:17 ./
drwxr-xr-x 11 www-data www-data 4096 Mar 14 16:17 ../
drwxrwxr-x 14 www-data www-data 4096 Dec 29 12:18 admin/
drwxrwxr-x 15 www-data www-data 4096 Dec 29 12:20 Arista_EOS_Tool/
drwxrwxr-x 14 www-data www-data 4096 Jan 29 21:21 examples/
-rw-rw-r--  1 www-data www-data   1 Dec 26 04:58 __init__.py
-rw-r--r--  1 www-data www-data 111 Dec 29 12:18 __init__.pyc
drwxrwxr-x 14 www-data www-data 4096 Dec 29 12:18 welcome/
```

Within each application, you can see the folders for model, controllers and views.

```
anees@ubuntu-web2py:/home/www-data/web2py/applications$ cd Arista_EOS_Tool/
anees@ubuntu-web2py:/home/www-data/web2py/applications/Arista_EOS_Tool$ ll
drwxrwxr-x 15 www-data www-data 4096 Dec 29 12:20 ./
drwxr-xr-x 8 www-data www-data 4096 Mar 14 16:17 ../
-rw-rw-r-- 1 www-data www-data 55 Dec 26 04:58 ABOUT
drwxr-xr-x 2 www-data www-data 4096 Dec 29 12:19 cache/
drwxrwxr-x 2 www-data www-data 4096 Dec 29 12:21 controllers/
drwxrwxr-x 2 www-data www-data 4096 Dec 26 04:58 cron/
drwxr-xr-x 2 www-data www-data 4096 May 19 09:22 databases/
drwxr-xr-x 2 www-data www-data 16384 May 19 09:40 errors/
-rw-rw-r-- 1 www-data www-data 1 Dec 26 04:58 __init__.py
-rw-r--r-- 1 www-data www-data 127 Dec 29 12:20 __init__.pyc
drwxrwxr-x 2 www-data www-data 4096 Dec 26 04:58 languages/
-rw-rw-r-- 1 www-data www-data 208 Dec 26 04:58 LICENSE
drwxrwxr-x 2 www-data www-data 4096 Jan 29 17:42 models/
drwxrwxr-x 2 www-data www-data 4096 Dec 29 12:20 modules/
drwxrwxr-x 2 www-data www-data 4096 Dec 26 04:58 private/
-rw-r--r-- 1 www-data www-data 45009 Jun 9 10:08 progress.log
-rw-rw-r-- 1 www-data www-data 1510 Dec 26 04:58 routes.example.py
drwxr-xr-x 20 www-data www-data 4096 Jun 6 08:17 sessions/
drwxrwxr-x 6 www-data www-data 4096 Jun 9 10:07 static/
drwxr-xr-x 2 www-data www-data 4096 Dec 29 12:19 uploads/
drwxrwxr-x 3 www-data www-data 4096 Jan 29 19:44 views/
```

You can find the default controller default.py under the controllers folder.

```
anees@ubuntu-web2py:/home/www-data/web2py/applications/Arista_EOS_Tool$ cd ↵
↵ controllers/

anees@ubuntu-web2py:/home/www-data/web2py/applications/Arista_EOS_Tool/controllers$ ll
drwxrwxr-x 2 www-data www-data 4096 Dec 29 12:21 ./
drwxrwxr-x 15 www-data www-data 4096 Dec 29 12:20 ../
-rw-rw-r-- 1 www-data www-data 25689 Dec 26 04:58 appadmin.py
-rw-rw-r-- 1 www-data www-data 33650 May 19 09:45 default.py
```

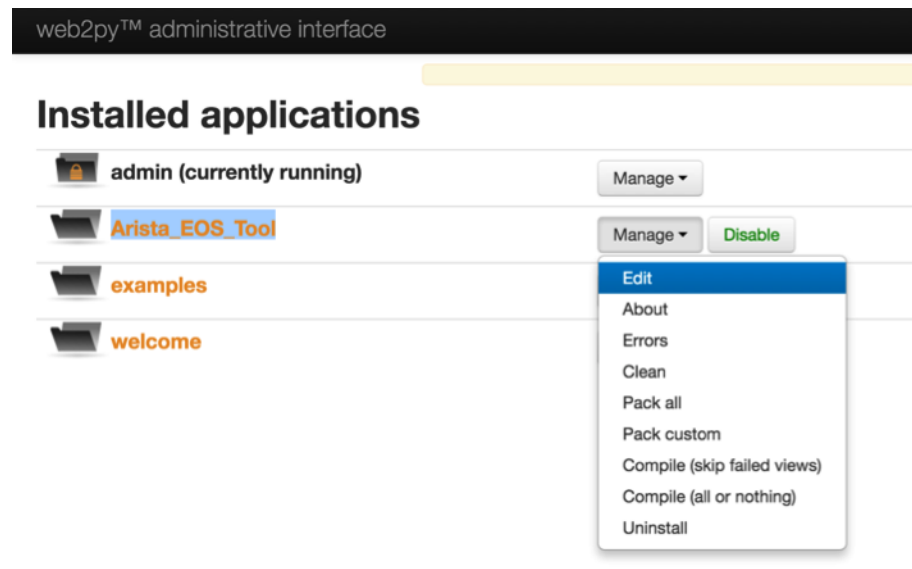
You can find the views under the “views/default” folder.

```
anees@ubuntu-web2py:/home/www-data/web2py/applications/Arista_EOS_Tool/controllers$ ↵
↵ cd ../views/default
anees@ubuntu-web2py:/home/www-data/web2py/applications/Arista_EOS_Tool/views/default$ ↵
↵ ll
-rw-rw-r-- 1 www-data www-data 1660 Jun 9 10:08 index.html
```

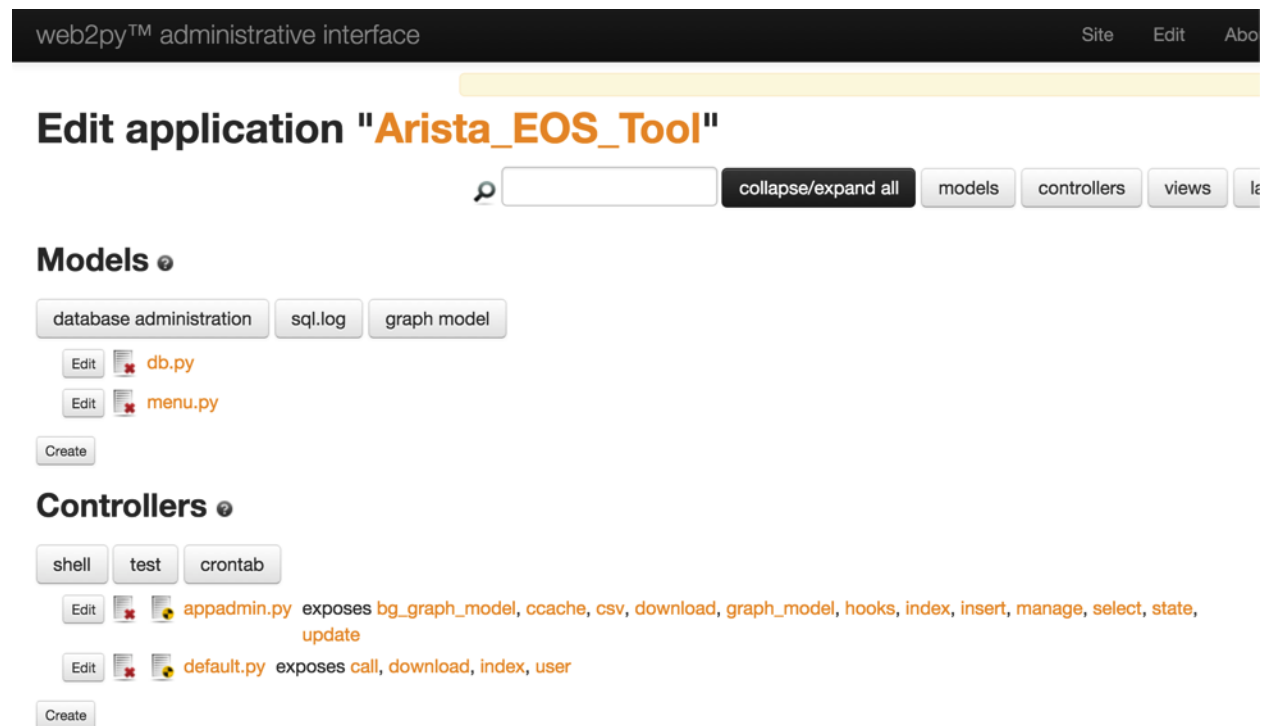
Edit the Application

We will remove the default scripts from the default controller and the view to start a clean empty application. Then from next chapter onwards, we will start populating our network use cases in this application.

Go to admin interface using the url <https://<web-server>/admin/default/index>. Click the “Manage” button and select Edit.



Click Edit which is right next to default.py controller.



You will see the default functions created by web2py. You should be able to see views (html files) for each of the function in this controller.

The screenshot shows the web2py administrative interface. At the top, there's a header with 'web2py™ administrative interface' and links for 'Site', 'Edit', and 'About'. Below the header, a sidebar on the left has a 'files toggle' button. The main area displays the 'default.py' file. Above the code editor, there's a status bar showing 'Save file:' with a file icon, 'Saved file hash: 23f8b97566757f32932f6ecb5fb5i', 'Last saved on: Sat Dec 26 04:58:48 2015', and a 'toggle breakpoint' button. Below this, it lists 'exposes: call, download, index, user' and 'edit views: index, user'. The code editor shows the following Python code:

```

1  # -*- coding: utf-8 -*-
2  # this file is released under public domain and you can use without limitations
3
4  #####
5  ## This is a sample controller
6  ## - index is the default action of any application
7  ## - user is required for authentication and authorization
8  ## - download is for downloading files uploaded in the db (does streaming)
9  #####
10
11 def index():
12     """
13     example action using the internationalization operator T and flash
14     rendered by views/default/index.html or views/generic.html
15
16     if you need a simple wiki simply replace the two lines below with:
17     return auth.wiki()
18     """
19     #return dict(dict(T=__('Hello World')))

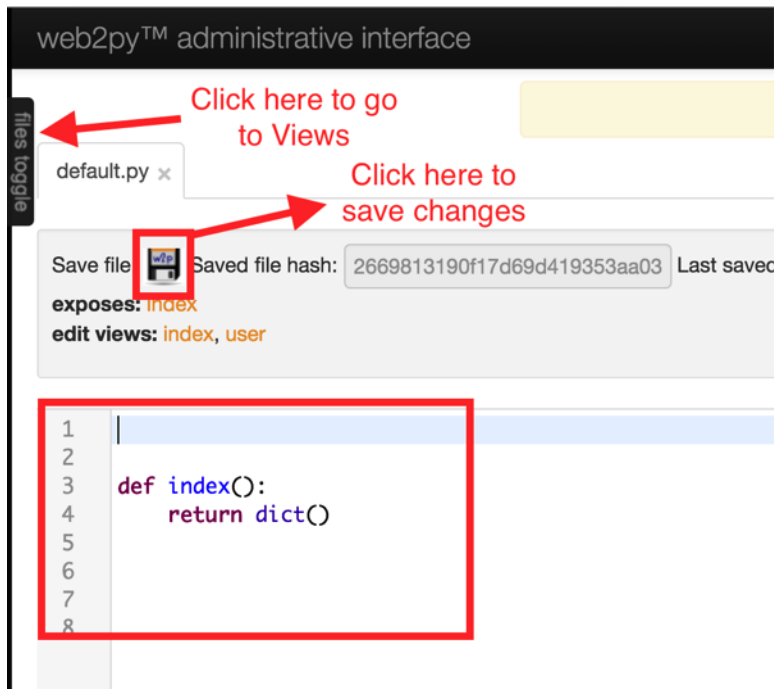
```

At the bottom of the editor, there are buttons for 'restore', 'currently saved or', 'revert', and 'to previous version.' Below the editor, there's a footer with 'TODO', 'Shortcuts', and 'Hooks'.

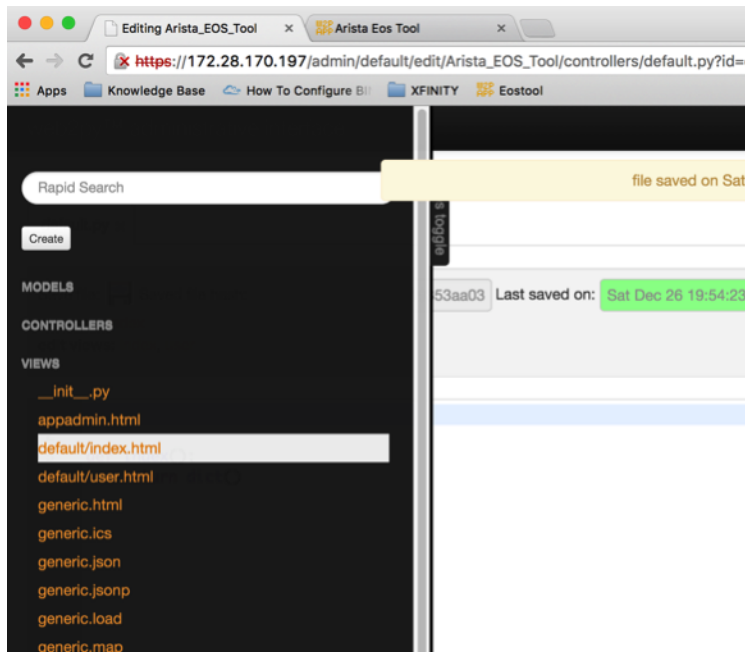
Remove all the scripts in the default.py and just enter the below Python script.

```
def index():
    return dict()
```

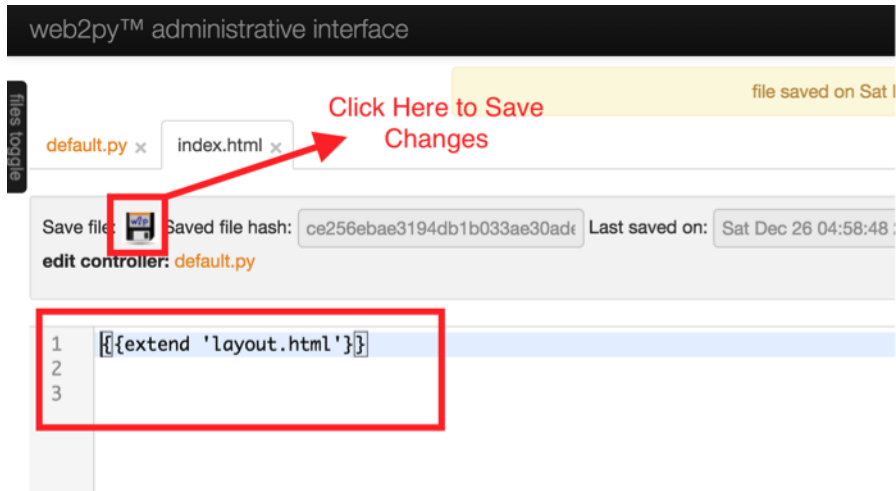
We have just created a function index() which does not have any logic and it just returns empty dictionary to the view. Save the script.



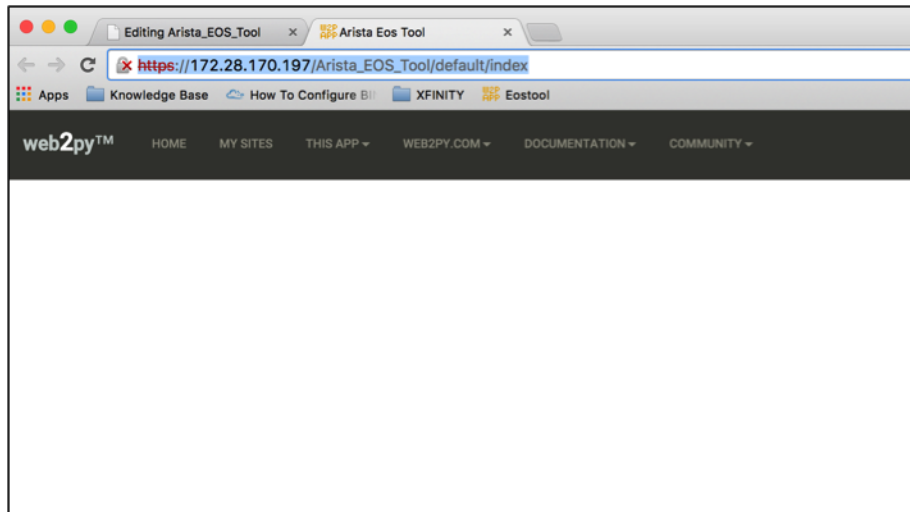
Now we will cleanup the view (Arista_EOS_Tool/views/default/index.html) for the index() function. Open the default view for index() function from administrative interface. On the left top, you will see files toggle → Views → default/index.html.



Delete the existing html scripts and just include the web2py's default layout. layout.html is hosted for every application you create through web2py. You can find this file inside the views folder of your application (Arista_EOS_Tool/views/).



Save the view and verify your blank application from browser http://<web-server>/Arista_EOS_Tool/default/index. The web page displays only the default layout (layout.html) which we included in the view.



We have successfully created a blank application. Let us move forward in hosting network operations use cases.

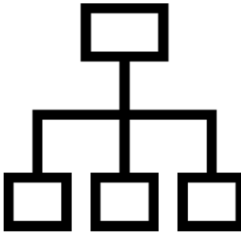
Chapter 8: Web2Py - Prepare the Tool

- *Add Switches*
 - *Algorithm*
 - *Develop Script*
 - * *Step 1: Create a New Function and a View for this task “Add Switches”*
 - * *Step 2: Display a form to receive input for username, password and IP addresses*
 - * *Step 3: Collect inventory and store it in a dictionary*
 - * *Step 4: Store the dictionary into a JSON file*
- *View Switches*
- *Categorize Switches*
 - *Algorithm*
 - *Develop Script*
 - * *Step 1: Create a New Function and View for this task “Categorize Switches”*
 - * *Step 2: Read the content of inventory.json and store it in a dictionary*
 - * *Step 3: Build a web2py form from the dictionary*
 - * *Step 4: Update site and role field in the dictionary*
 - * *Step 5: Write the dictionary in the inventory.json file*
- *Summary*

In this chapter, we will create functions and views for the tasks in the “Prepare the Tool” category. The purpose of these tasks in “Prepare the Tool” category is to provide an option to add the switches in the network manually and maintain the inventory. We will also provide an option to let the users to categorize the switches based on the location and role. Once the users added the switches in the tool, then they can run any operational tasks against those switches.

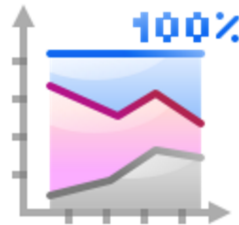
The user will be able to run the tasks on all the switches or on the switches that belong to a specific site and/or on the switches with a specific role.

Arista Professional Services



Prepare the Tool

1. Add Switches
2. Categorize Switches
3. View Switches



Capacity Planning

1. Port Capacity
2. Hardware Tables Capacity (EMT, LPM, and TCAM)



Network-wide Troubleshooting

1. Packet Drops
2. Control Plane Issues (T2 Only)
3. Last 10 Sev 1-3 Log messages

We are going to create three tasks (Add Switches, Categorize Switches and View Switches) in this section. Each task will have one or more functions in the default controller and a corresponding view (.html file).

Add Switches

“Add Switches” task allows the users to manually add the IP address of the switches into the tool. The tool is going to present a form to the end users where they can enter username, password and the IP address of the switches. Then the task is going to collect some of the basic information about the switches using pyeapi. Finally the task will store the inventory of these switches locally in the server. The task will not save the username and the password anywhere in the server.

We are going to write a function `add_inventory` in the default controller of our application. Since this will be our first program using web2py, we are going to spend more time in understanding how web2py works. First we will write an algorithm for this task.

Algorithm

We are going to collect the information and report it in the following format.

```
Inventory = {"<switch_IP_Address>": {"Hostname": "<switch_host_name>",
                                     "Model": "<switch_model>",
                                     "Version": "<switch_EOS_Version>",
                                     "Serial Number": "<Serial_Number>",
                                     "Site": "none",
                                     "Role": "none"
                                    }
            }
```

The following Arista EOS commands are used to collect the above information.

```
23sw35#show hostname | json
{
  "fqdn": "23sw35",
  "hostname": "23sw35"
}
```

```
23sw35#show version | json
{
  "modelName": "DCS-7250QX-64-F",
  "internalVersion": "4.15.5M-3054042.4155M",
  "systemMacAddress": "00:1c:73:5d:13:e9",
  "serialNumber": "JPE14300059",
  "memTotal": 8069500,
  "bootupTimestamp": 1467299199.33,
  "memFree": 4438208,
  "version": "4.15.5M",
  "architecture": "i386",
  "internalBuildId": "43b3ce87-6eeb-48ce-bd2a-48e98def005a",
  "hardwareRevision": "01.03"
}
```

Once we collect the data in a dictionary, it is easy to display the content of the dictionary from the view and store it in a file in json format locally in the server.

1. Create a New Function and a View for this task “Add Switches”
2. Display a form to input username, password and IP address of the switches.
3. Collect the inventory from the switches using pyeapi and store it in a dictionary.
4. Store the dictionary in a json file called inventory.json. Save this file in the /home/www-data/web2py/applications/Arista_EOS_Tool/databases folder.

Develop Script





Step 1: Create a New Function and a View for this task “Add Switches”

Go to admin interface using the url <https://<web-server>/admin/default/index>

Arista_EOS_Tool: Manage -> Edit

web2py™ administrative interface

Installed applications


 admin (currently running)	Manage ▾
 Arista_EOS_Tool	Manage ▾ Disable
 examples	
 welcome	

Edit
About
Errors
Clean
Pack all
Pack custom
Compile (skip failed views)
Compile (all or nothing)
Uninstall

Controllers: default.py → Edit


web2py™ administrative interface Site Edit Abo


Edit application "Arista_EOS_Tool"

 collapse/expand all models controllers views le

Models

database administration sql.log graph model


Edit  **db.py**


Edit  **menu.py**

Create

Controllers

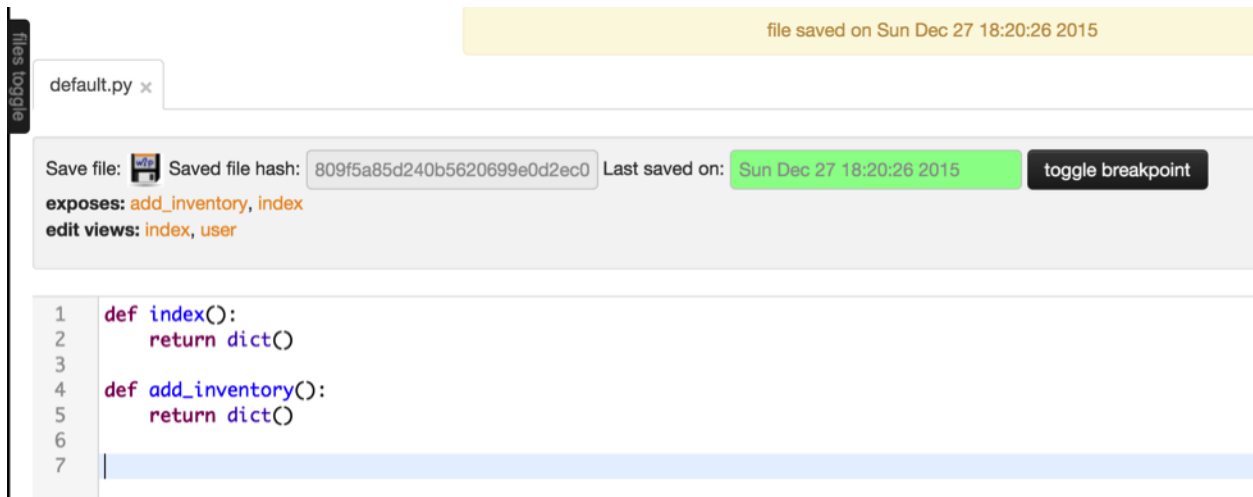
shell test crontab

Edit  **appadmin.py** exposes **bg_graph_model**, **ccache**, **csv**, **download**, **graph_model**, **hooks**, **index**, **insert**, **manage**, **select**, **state**, **update**

Edit  **default.py** exposes **call**, **download**, **index**, **user**

Create

Create a new function `add_inventory()`

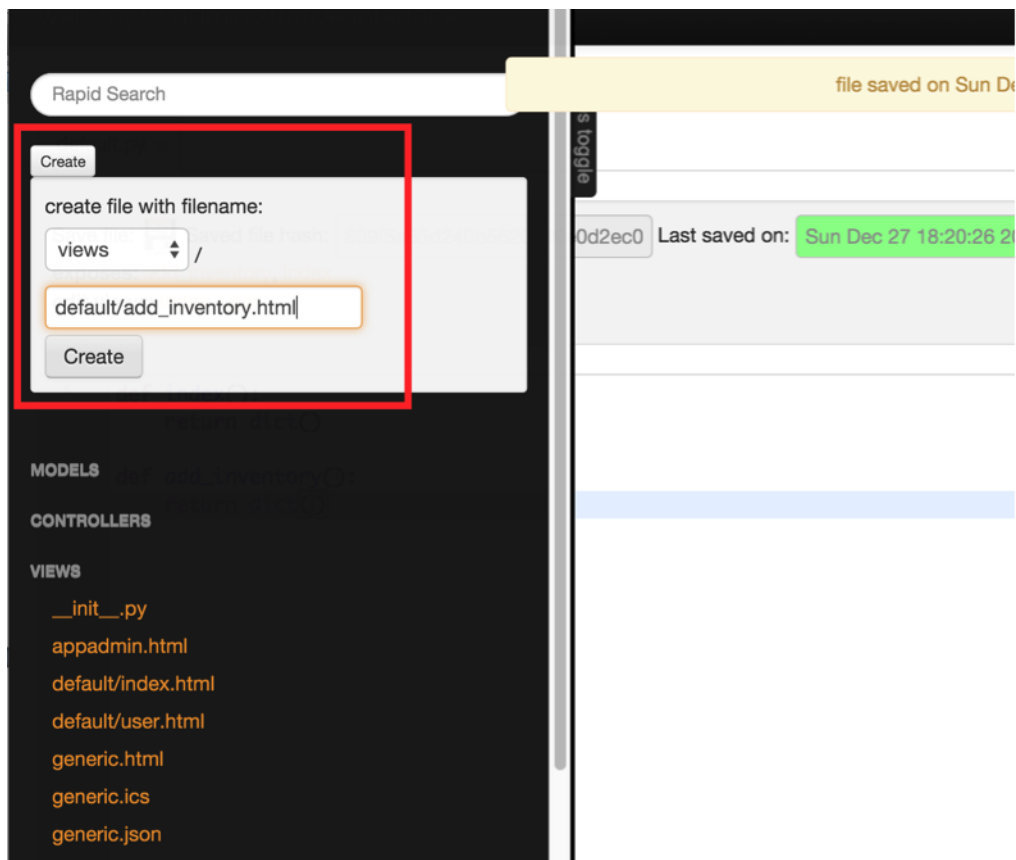


Create a View for the function add_inventory

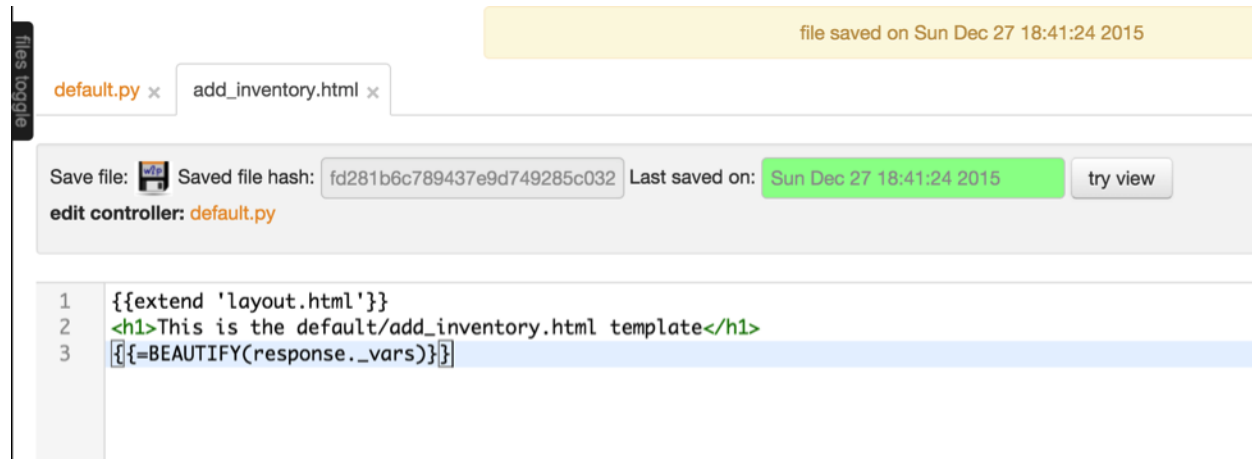
Click the “files toggle” on the top left

Click Create and select views from the drop down window

provide the file name with path -> default/add_inventory.html



We are going to keep the default content inside the view. Save this file.



You can verify the new function using the URL https://<web-server>/Arista_EOS_Tool/default/add_inventory. Since the function `add_inventory` is blank and it returns empty dictionary to the view. The view shows the default layout which shows the default web2py menu bar in the top of the screen and the title “This is the default/add_inventory.html template”.

Step 2: Display a form to receive input for username, password and IP addresses

There are few different ways to build forms in web2py. We are going to create a form using web2py’s `SQLFORM.factory`. We will define a form using `SQLFORM.factory` and assign it to a variable called `form`. Then we will return this variable to view using “`return dict(form=form)`”.

As you can see, there are three fields defined for our form. The first string inside each `Field()` entry is the name of the variable in which the values the user enters will be stored. This should be unique within the form. Rest of the strings within the `Field()` are optional.

Edit the `add_inventory` function in the default controller.

```

def add_inventory():
    form = SQLFORM.factory(
        Field('username', requires=IS_NOT_EMPTY()),
        Field('password', 'password', requires=IS_NOT_EMPTY()),
        Field('switchip', 'text', label="Switch IPs")

    return dict(form=form)

```

We don’t have to update the view since we are going to display all variables from the function using the statement “`{{=BEAUTIFY(response._vars)}}`”. We are going to discuss more about views in chapter 11.

Verify the updated function using the same URL https://<web-server>/Arista_EOS_Tool/default/add_inventory.

As you can see the field for switches is larger than for username and password. This is because we declared this field as 'text' when we define the form. Similarly, when you enter the field for password, it won't display the content of the password. This is because we declare this field as "password" when we define the form.

You can change the display of the field different than the variable name by using label. You can define the fields as mandatory using `requires=IS_NOT_EMPTY()`.

You can refer the "Forms and validators" chapter in the [web2py documentation](#) to learn more about web2py forms.

Step 3: Collect inventory and store it in a dictionary

Once the user enters the username, password, IP addresses and submit the form, the script should initiate the `pyeapi` call and collect the inventory from the switches. The inventory will be stored in a dictionary and displayed to the end user by returning the dictionary to the view.

First we will understand how to accept the values of the form variables from the default controller. So let us update our `add_inventory()` function to display the value of the IP addresses after the user clicks the submit button.

```
def add_inventory():
    # Display form to input Username, Password and Switch IP addresses
    form = SQLFORM.factory(
        Field('username', requires=IS_NOT_EMPTY()),
        Field('password', 'password', requires=IS_NOT_EMPTY()),
        Field('switchip', 'text', label="Switch IPs"))

    # if the form is accepted, collect the information from the switches
    if form.process().accepted:
        # Initiate inventory with blank dictionary
        inventory = {}

        # Since switch IPs are input as text with multiple IPs one per line,
        # We will convert the text into List with the list of switch IP addresses
```

```

switchip_list = form.vars.switchip.split("\n")

# For each IP in the list switchip_List, collect the inventory
for each_switch_ip in switchip_list:
    # For each switch IP, create empty directory with key as switch IP
    inventory[each_switch_ip.strip()] = {}

# Return the inventory to View
return dict(inventory=inventory)

# Initially form will be returned to the view.
return dict(form=form)

```

Save the default.py and verify your script using the URL https://<web-server>/Arista_EOS_Tool/default/add_inventory. Before that let us update the view (add_inventory.html) to display the title as “Add Switches”.

```

{{extend 'layout.html'}}
<h1>Add Switches</h1>
{{=BEAUTIFY(response._vars)}}

```



When you enter https://<web-server>/Arista_EOS_Tool/default/add_inventory, add_inventory() function executes. First, the form variable is assigned with the fields defined using SQLFORM.factory() method. When it executes “if form.process().accepted:” statement, it bypasses the if clause since the form has not been submitted yet. Then the last statement of the add_inventory() function returns the form variable to the view add_inventory.html.

Once you enter the username, password, switch IPs and submit, “if form.process().accepted:” clause is executed. Since we define the variable inventory and return this dictionary to the view within the “if form.process().accepted:” clause, we are seeing the content of the dictionary “inventory” on the web page. The values of the form fields are assigned internally by form.vars.<variable_name_defined_in_the_field> (for example form.vars.username, form.vars.password and form.vars.switchip).

Now we understand how to use web2py forms and display data using view, let us update the add_inventory() function to collect the inventory of the switches and store it in the dictionary.


```

import pyeapi

# Default Index for Home page of this tool
def index():
    return dict()

# Prepare the Tool: Add Switches
def add_inventory():
    # Display form to input Username, Password and Switch IP addresses
    form = SQLFORM.factory(
        Field('username', requires=IS_NOT_EMPTY()),
        Field('password', 'password', requires=IS_NOT_EMPTY()),
        Field('switchip', 'text', label="Switch IPs"))

    # if the form is accepted, collect the information from the switches
    if form.process().accepted:
        # Initiate inventory & error with blank dictionary
        inventory = {}
        errors = {}

        # Convert Switch IPs field from string to list
        switchip_list = form.vars.switchip.split("\n")

        # For each IP in the List switchip_List, apply your logic
        for each_switch_ip in switchip_list:
            # For each switch IP, create empty directory with key as switch IP
            inventory[each_switch_ip.strip()] = {}

            # Connect to Switches and Collect Inventory
            try:
                node = pyeapi.connect(transport='https', host=each_switch_ip.strip(),
↪username=form.vars.username, password=form.vars.password, port=None)

                # Collect the inventory
                show_hostname = node.execute(["show hostname"])
                hostname = str(show_hostname["result"][0]["hostname"])

                show_inventory = node.execute(["show inventory"])
                model = str(show_inventory["result"][0]["systemInformation"]["name"])

                show_version = node.execute(["show version"])
                version = str(show_version["result"][0]["version"])
                serialnumber = str(show_version["result"][0]["serialNumber"])

                # Save the collected data in the inventory dictionary
                inventory[each_switch_ip.strip()] = {"hostname": hostname, "model":
↪model, "serialnumber": serialnumber, "version": version, "site": "none", "role":
↪"none"}

            except pyeapi.eapilib.ConnectionError:
                errors[each_switch_ip.strip()] = "ConnectionError: unable to connect
↪to eAPI"

            except pyeapi.eapilib.CommandError:
                errors[each_switch_ip.strip()] = "CommandError: Check your EOS
↪command syntax"

        # Return the inventory to View

```

```
    return dict(errors=errors, inventory=inventory)

# Return form to view.
return dict(form=form)
```

Save the config and verify the result.

Add Switches

errors :172.28.170.98:ConnectionError: unable to connect to eAPI

inventory:172.28.170.114:hostname :22sw35
model :DCS-7250QX-64
role :none
serialnumber:JPE14421537
site :none
version :4.15.3F
172.28.170.115:hostname :22sw37
model :DCS-7250QX-64
role :none
serialnumber:JPE14402468
site :none
version :4.15.3F
172.28.170.97 :hostname :dc-spine-3
model :DCS-7050SX-128
role :none
serialnumber:JPE14080457
site :none
version :4.15.3F
172.28.170.98 :

Step 4: Store the dictionary into a JSON file

Store the dictionary in JSON format and save under the folder /home/www-data/web2py/applications/Arista_EOS_Tool/databases/. The reason for storing the data in a file is that we will reuse this data (Switch IP addresses, site and role) by all the uses cases created in this tool.

First create a blank inventory.json file in the server.

```
anees@ubuntu-web2py:~$ cd /home/www-data/web2py/applications/Arista_EOS_Tool/
↳ databases/

anees@ubuntu-web2py:/home/www-data/web2py/applications/Arista_EOS_Tool/databases$
↳ sudo sh -c "echo {} > inventory.json"

anees@ubuntu-web2py:/home/www-data/web2py/applications/Arista_EOS_Tool/databases$
↳ sudo chown -R www-data:www-data inventory.json
```

Update the script to store the content of the dictionary (inventory) into this file.

```

import pyeapi
import json

# Define inventory file
file_path = "/home/www-data/web2py/applications/Arista_EOS_Tool/databases/"
file_inventory = "inventory.json"
file = file_path + file_inventory

# Default Index for Home page of this tool
def index():
    return dict()

# Prepare the Tool: Add Switches
def add_inventory():
    # Display form to input Username, Password and Switch IP addresses
    form = SQLFORM.factory(
        Field('username', requires=IS_NOT_EMPTY()),
        Field('password', 'password', requires=IS_NOT_EMPTY()),
        Field('switchip', 'text', label="Switch IPs"))

    # if the form is accepted, collect the information from the switches
    if form.process().accepted:
        # Initiate inventory & error with blank dictionary
        inventory = {}
        errors = {}

        # Convert Switch IPs field from string to list
        switchip_list = form.vars.switchip.split("\n")

        # For each IP in the List switchip_List, apply your logic
        for each_switch_ip in switchip_list:
            # For each switch IP, create empty directory with key as switch IP
            inventory[each_switch_ip.strip()] = {}

            # Connect to Switches and Collect Inventory
            try:
                node = pyeapi.connect(transport='https', host=each_switch_ip.strip(),
↪username=form.vars.username, password=form.vars.password, port=None)

                # Collect the inventory
                show_hostname = node.execute(["show hostname"])
                hostname = str(show_hostname["result"][0]["hostname"])

                show_inventory = node.execute(["show inventory"])
                model = str(show_inventory["result"][0]["systemInformation"]["name"])

                show_version = node.execute(["show version"])
                version = str(show_version["result"][0]["version"])
                serialnumber = str(show_version["result"][0]["serialNumber"])

                # Save the collected data in the inventory dictionary
                inventory[each_switch_ip.strip()] = {"hostname": hostname, "model":
↪model, "serialnumber": serialnumber, "version": version, "site": "none", "role":
↪"none"}

            except pyeapi.eapilib.ConnectionError:
                errors[each_switch_ip.strip()] = "ConnectionError: unable to connect
↪to eAPI"

```

```
except pyeapi.eapilib.CommandError:
    errors[each_switch_ip.strip()] = "CommandError: Check your EOS_
↪command syntax"

# Store the dictionary "inventory" in the json file
with open(file) as readfile:
    current_inventory = json.load(readfile)
    current_inventory.update(inventory)

with open(file, 'w') as writefile:
    json.dump(current_inventory, writefile)

# Return the inventory to View
return dict(errors=errors, inventory=inventory)

# Return form to view.
return dict(form=form)
```

Verify your script using the URL https://<web-server>/Arista_EOS_Tool/default/add_inventory. The result will be displayed on the web page as before. You can also check the content of the inventory.json from the ubuntu server.

```
anees@ubuntu-web2py:~$ cd /home/www-data/web2py/applications/Arista_EOS_Tool/
↪databases/
anees@ubuntu-web2py:/home/www-data/web2py/applications/Arista_EOS_Tool/databases$ cat_
↪inventory.json
{"172.28.170.98": {"serialnumber": "JPE14080459", "hostname": "22sw4", "site": "none",
↪ "version": "4.15.3F", "role": "none", "model": "DCS-7050SX-128"}, "172.28.170.97":
↪{"serialnumber": "JPE14080457", "hostname": "22sw2", "site": "none", "version": "4.
↪15.3F", "role": "none", "model": "DCS-7050SX-128"}, "172.28.170.114": {"serialnumber
↪": "JPE14421537", "hostname": "22sw35", "site": "none", "version": "4.15.3F", "role
↪": "none", "model": "DCS-7250QX-64"}, "172.28.170.115": {"serialnumber":
↪"JPE14402468", "hostname": "22sw37", "site": "none", "version": "4.15.3F", "role":
↪"none", "model": "DCS-7250QX-64"}}anees@ubuntu-web2py:/home/www-data/web2py/
↪applications/Arista_EOS_Tool/databases$
```

View Switches

“View Switches” task allows the users to view the inventory of the switches stored in the tool. We will write a new function called `view_inventory()` in the `default.py` to show the content of the `inventory.json` file. The logic is very simple. Read the json file into a dictionary and return that dictionary to the view.

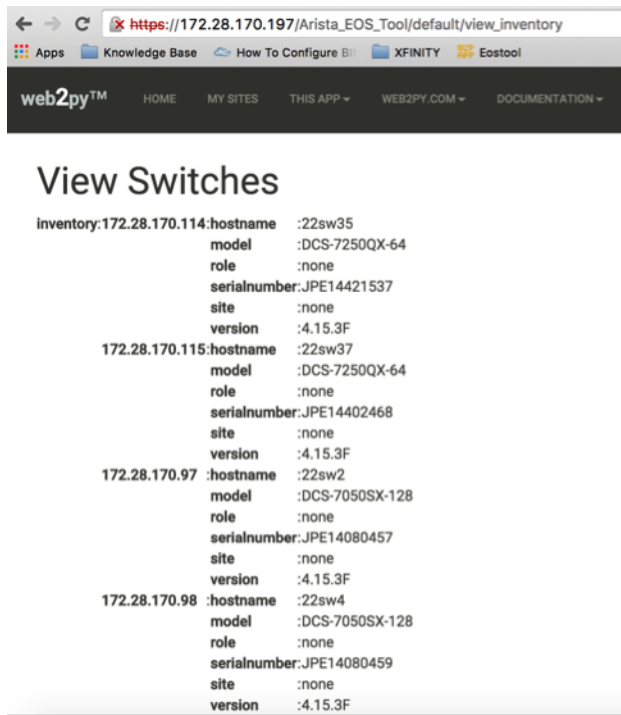
Edit the `default.py` file of the web2py application `Arista_EOS_Tool` and add this `view_inventory()` function.

```
def view_inventory():
    with open(file) as readfile:
        inventory = json.load(readfile)
    return dict(inventory=inventory)
```

Create a new web2py view `view_inventory.html` (`default/view_inventory.html`) for this new function using web2py editor.

```
{{extend 'layout.html'}}
<h1>View Switches</h1>
{{=BEAUTIFY(response._vars)}}
```

Verify the new function using the URL https://<web-server>/Arista_EOS_Tool/default/view_inventory. You should see the result as below:



Categorize Switches

After we added the switches manually, we are going to classify the switches with based on the location and role. This gives the user an option to run any scripts in this tool against specific set of switches. For example, we will be writing a script to find unused ports. The user may want to run this script only on all leaf switches from a specific data center. Let us write an algorithm for this task.

Algorithm

When a user adds switches using the add switches task, the script will save the inventory information in the inventory.json file. When it stores for the first time, it assigns the value "None" to the fields site and role.

In this categorize switches task, we are going to read and display the switches and its corresponding site and role in a web2py form. This will allow the users to assign site and role for all the switches in the inventory.json file.

1. Create a New Function and a View for this task "Categorize Switches"
2. Read the content of inventory.json file and assign it to a dictionary variable called inventory
3. Build a web2py form with the fields switch's hostname, site and role from the content of inventory dictionary.
4. Once the user submit the form with the updated site and role fields, update the dictionary variable "inventory"
5. Write the dictionary inventory to the inventory.json file.

Develop Script

Step 1: Create a New Function and View for this task “Categorize Switches”

Edit the default.py file of the web2py application Arista_EOS_Tool and add this categorize_inventory() function.

```
def categorize_inventory():  
    return dict()
```

Create a new web2py view categorize_inventory.html (default/categorize_inventory.html) for this new function using web2py editor.

```
{{extend 'layout.html'}}  
<h1>Categorize Switches</h1>  
{{=BEAUTIFY(response._vars)}}
```

Step 2: Read the content of inventory.json and store it in a dictionary

Update the categorize_inventory() function to read the inventory.json file.

```
def categorize_inventory():  
    with open(file) as readfile:  
        inventory = json.load(readfile)  
    return dict()
```

Step 3: Build a web2py form from the dictionary

This is an interesting step in this task. We have to build a form with the fields hostname, site and role from the content of inventory variable. First, how did we create a form previously in this chapter?

```
form = SQLFORM.factory(  
    Field('username', requires=IS_NOT_EMPTY()),  
    Field('password', 'password', requires=IS_NOT_EMPTY()),  
    Field('switchip', 'text', label="Switch IPs"))
```

In the above form, we know what fields need to be created. But in this use case, We have to create fields based on the content of inventory.json file. Our goal is to display field for each switch in the inventory.json file. Well actually we have to create two fields (Site and Role) for each switch in the inventory.json file. So the number of fields depends on the number of switches in the inventory.json file.

Web2py form allows to create a form using a list of fields. You can create a list with the Field() objects and then you can create SQLFORM.factory using that list.

- We are going to create two fields site and role for each switch.

```
Field("site", label=str(hostname)+" Site", default=site)  
Field("role", label=str(hostname)+" Role", default=role)
```

“Site” and “Role” are the variable names for the data that the user is going to input. We use switch hostname as a label so that the user can classify the switch properly. We don’t want to show blank field for Site and Role. We want to populate the current Site and Role.

- We have to populate the above two fields for all the switches in the inventory file. So we need somehow to create unique variable names for “Site” and “Role.” We can use integers starting 1. For example, site1, role1 for switch1, site2, role2 for switch2, etc. We will put all these fields in a list.

```

# Define an empty list for fields
fields = []

# Define a variable for integers which we will use to create unique variable names
field_variable = 0

# For each switch, create fields and add it to the list field
for each_switch_ip in inventory.keys():
    field_variable += 1

    # We will grab hostname, site and role from current inventory
    # We will use these in the Field variable
    hostname = inventory[each_switch_ip]["hostname"]
    site = inventory[each_switch_ip]["site"]
    role = inventory[each_switch_ip]["role"]

    # create variables and append to the list
    fields.append(Field("site"+str(field_variable), label=str(hostname)+" Site",
↳default=site))
    fields.append(Field("role"+str(field_variable), label=str(hostname)+" Role",
↳default=role))

```

- Create a form using the fields list

```
form = SQLFORM.factory(*fields)
```

Let us update the `categorize_inventory()` function with this new SQLFORM.

```

def categorize_inventory():
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define an empty list for fields
    fields = []

    # Define a variable for integers which we will use to create unique variable names
    field_variable = 0

    # For each switch, create fields and add it to the list field
    for each_switch_ip in inventory.keys():
        field_variable += 1

        # We will grab hostname, site and role from current inventory
        # We will use these in the Field variable
        hostname = inventory[each_switch_ip]["hostname"]
        site = inventory[each_switch_ip]["site"]
        role = inventory[each_switch_ip]["role"]

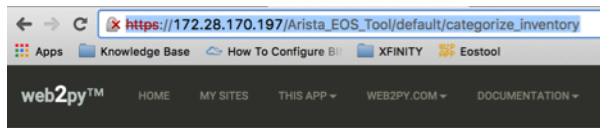
        # create variables and append to the list
        fields.append(Field("site"+str(field_variable), label=str(hostname)+" Site",
↳default=site))
        fields.append(Field("role"+str(field_variable), label=str(hostname)+" Role",
↳default=role))

    # Create a form using the list fields
    form=SQLFORM.factory(*fields)

    return dict(form=form)

```

Test the script using the URL https://<web-server>/Arista_EOS_Tool/default/categorize_inventory



Categorize Switches

form:

22sw4 Site	<input type="text" value="none"/>
22sw4 Role	<input type="text" value="none"/>
22sw37 Site	<input type="text" value="none"/>
22sw37 Role	<input type="text" value="none"/>
22sw35 Site	<input type="text" value="none"/>
22sw35 Role	<input type="text" value="none"/>

Step 4: Update site and role field in the dictionary

As we have unique variable names for Site and Role for each switch in inventory.json file, we will use the similar “for loop” statement to update the dictionary “inventory.”

```
def categorize_inventory():
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define an empty list for fields
    fields = []

    # Define a variable for integers which we will use to create unique variable names
    field_variable = 0

    # For each switch, create fields and add it to the list field
    for each_switch_ip in inventory.keys():
        field_variable += 1

        # We will grab hostname, site and role from current inventory
        # We will use these in the Field variable
        hostname = inventory[each_switch_ip]["hostname"]
        site = inventory[each_switch_ip]["site"]
        role = inventory[each_switch_ip]["role"]

        # create variables and append to the list
        fields.append(Field("site"+str(field_variable), label=str(hostname)+" Site",
↪default=site))
        fields.append(Field("role"+str(field_variable), label=str(hostname)+" Role",
↪default=role))

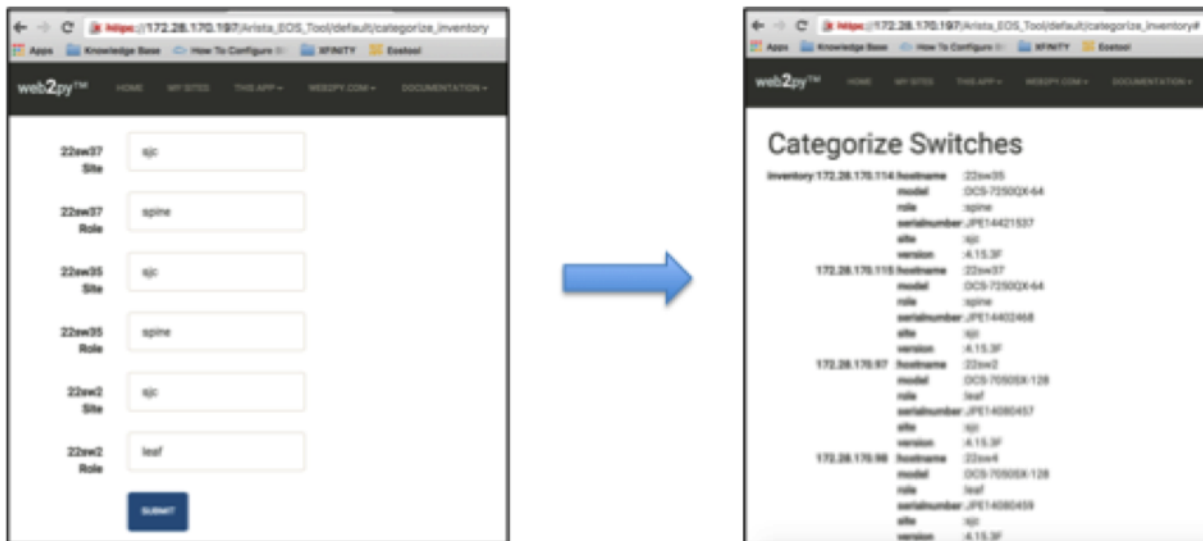
    # Create a form using the list fields
    form=SQLFORM.factory(*fields)
```



```
# Update dictionary based on the user input
if form.process().accepted:
    field_variable = 0
    for each_switch_ip in inventory.keys():
        field_variable += 1
        inventory[each_switch_ip]["site"] = form.vars["site"+str(field_variable)]
        inventory[each_switch_ip]["role"] = form.vars["role"+str(field_variable)]
    return dict(inventory=inventory)

return dict(form=form)
```

Test the script using the URL https://<web-server>/Arista_EOS_Tool/default/categorize_inventory



Step 5: Write the dictionary in the inventory.json file

We will update the script to write the dictionary into the inventory.json file.

```
def categorize_inventory():
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define an empty list for fields
    fields = []

    # Define a variable for integers which we will use to create unique variable names
    field_variable = 0

    # For each switch, create fields and add it to the list field
    for each_switch_ip in inventory.keys():
        field_variable += 1

        # We will grab hostname, site and role from current inventory
        # We will use these in the Field variable
        hostname = inventory[each_switch_ip]["hostname"]
        site = inventory[each_switch_ip]["site"]
        role = inventory[each_switch_ip]["role"]
```

```
    # create variables and append to the list
    fields.append(Field("site"+str(field_variable), label=str(hostname)+" Site",
↪default=site))
    fields.append(Field("role"+str(field_variable), label=str(hostname)+" Role",
↪default=role))

    # Create a form using the list fields
    form=SQLFORM.factory(*fields)

    # Update dictionary based on the user input
    if form.process().accepted:
        field_variable = 0
        for each_switch_ip in inventory.keys():
            field_variable += 1
            inventory[each_switch_ip]["site"] = form.vars["site"+str(field_variable)]
            inventory[each_switch_ip]["role"] = form.vars["role"+str(field_variable)]

    # Store the dictionary "inventory" in the json file
    with open(file, 'w') as writefile:
        json.dump(inventory, writefile)

    return dict(inventory=inventory)

return dict(form=form)
```

Test the script using the URL https://<web-server>/Arista_EOS_Tool/default/categorize_inventory

You can use the `view_inventory()` function to verify whether the data is updated in the `inventory.json`. Test `view_inventory()` using https://<web-server>/Arista_EOS_Tool/default/view_inventory.

View Switches

```
inventory:172.28.170.114:hostname :22sw35
                        model      :DCS-7250QX-64
                        role        :spine
                        serialnumber:JPE14421537
                        site         :sjc
                        version      :4.15.3F
172.28.170.115:hostname :22sw37
                        model      :DCS-7250QX-64
                        role        :spine
                        serialnumber:JPE14402468
                        site         :sjc
                        version      :4.15.3F
172.28.170.97 :hostname :22sw2
                        model      :DCS-7050SX-128
                        role        :leaf
                        serialnumber:JPE14080457
                        site         :sjc
                        version      :4.15.3F
172.28.170.98 :hostname :22sw4
                        model      :DCS-7050SX-128
                        role        :leaf
                        serialnumber:JPE14080459
                        site         :sjc
                        version      :4.15.3F
```

Summary

We have completed three use cases under “Preparing the Tools” section. You can add more use cases as per your requirement. For example, you can create a use case for update inventory. If any of the inventory data such as hostname, software version or serial number (RMA), is changed, and if you need to delete any of the switches from the inventory, you can accomplish those with this use case.

Each use case is a separate function within the default controller “default.py” and we access each of these use cases with the URL https://<web-server>/Arista_EOS_Tool/default/<Name-of-the-function>.

Later in this book, we will create a home page with the links to all of these functions (use cases). The home page will be accessible using the URL https://<web-server>/Arista_EOS_Tool/default/index.

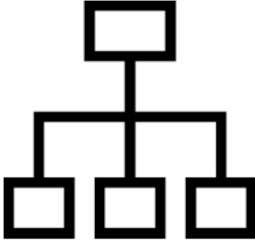
Chapter 9: Web2Py - Troubleshooting Use Cases

- *Data Plane Packet Drops*
 - *Algorithm*
 - *Develop Script*
 - * *Step 1: Create a New Function and a View for this task “Packet Drops”*
 - * *Step 2: Display a form to get username, password, site and role*
 - * *Step 3: Build Switch IP Address list*
 - * *Step 4: Reuse the packet drops Python script*
 - *Script Enhancements*
 - * *Step 1: Create a function to create a web2py form*
 - * *Step 2: Create a function to derive the list of switches*
 - * *Step 3: Create a function for discovering interface drops*
 - *Final Script*
- *Control Plane Drops*
- *Last 10 Sev 1-3 Log Messages*
 - *Algorithm*
 - *Develop Script*
 - * *Step 1: Create a New Function and a View for this task “show logs”*
 - * *Step 2: Use the eos_form() and switches_list() function to display form*
 - * *Step 3: Write the core logic of the script*
 - *Final Script*

In this chapter, we will create functions and views for the tasks in the “Network-wide Troubleshooting” category. The purpose of these tasks is to perform some of the common network troubleshooting steps across the network using the tool.

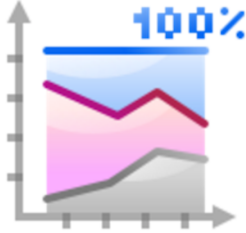
Arista HOME ARISTA LINKS ▾ OPERATIONS RUN BOOK

Arista Professional Services




Prepare the Tool

1. Add Switches
2. Categorize Switches
3. View Switches



Capacity Planning

1. Port Capacity
2. Hardware Tables Capacity (EMT, LPM, and TCAM)



Network-wide Troubleshooting

1. Packet Drops
2. Control Plane Issues (T2 Only)
3. Last 10 Sev 1-3 Log messages

We are going to create three tasks (Data Plane Packet Drops, Control Plane Drops and Last 10 Sev1-3 log messages) in this section. Each task will have one or more functions in the default controller and a corresponding view (.html file).

Data Plane Packet Drops

We are going to write a function `packet_drops()` in the default controller of our web2py application `Arista_EOS_tool`. We have already written a Python script to discover network-wide packet drops in the earlier chapter in this book. We are going to reuse that script here. The core logic is going to be the same. The only change here is to receive the input with web2py form. The input fields are username, password, site and role. We are going to create a drop-down menu to show the sites and roles so that the users don't have to type them.

form : Username:

Password:

Network:

✓
sjc
All

Role:

Algorithm

We are going to use the below algorithm to write the script.

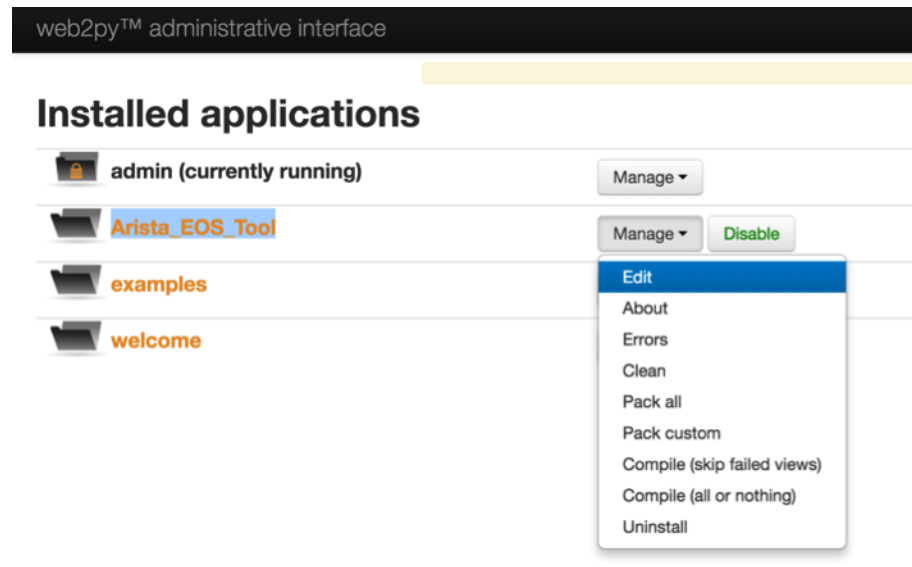
1. Create a New Function and a View for this task “Packet Drops”.
2. Display a form to get username, password, site and role.
3. Build Switch IP Address list.
4. Reuse the packet drops Python script and returns the result to the view.

Develop Script

Step 1: Create a New Function and a View for this task “Packet Drops”

Go to admin interface using the url <https://<web-server>/admin/default/index>


Arista_EOS_Tool: Manage -> Edit



Controllers: default.py -> Edit


web2py™ administrative interface Site Edit Abo


Edit application "Arista_EOS_Tool"

 collapse/expand all models controllers views le

Models

database administration sql.log graph model



Edit  db.py



Edit  menu.py

Create

Controllers

shell test crontab

Edit   appadmin.py exposes bg_graph_model, ccache, csv, download, graph_model, hooks, index, insert, manage, select, state, update

Edit   default.py exposes call, download, index, user

Create

Create a new function packet_drops()

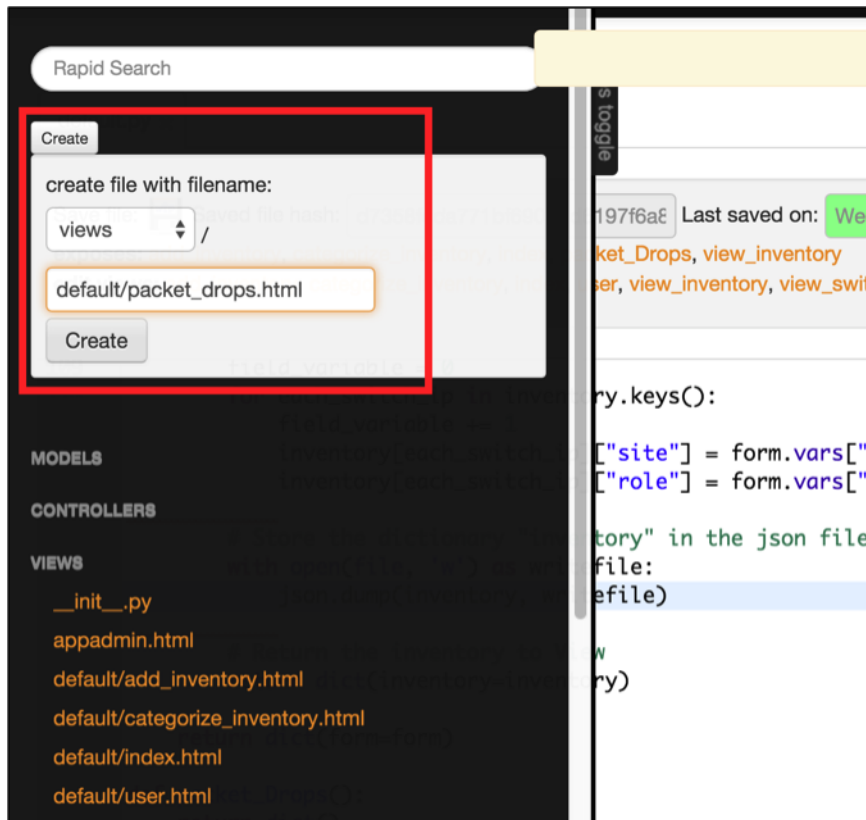
```
def packet_drops() :
    return dict()
```

Create a View for the function packet_drops

Click the “files toggle” on the top left

Click Create and select views from the drop down window

Provide the file name with path → default/packet_drops.html



We are going to create a default view with the header “Packet Drops”. Save this file.

```
{{extend 'layout.html'}}
<h1>Packet Drops</h1>
{{=BEAUTIFY(response._vars)}}
```

Step 2: Display a form to get username, password, site and role

We know how to create a form with the static fields such as username and password using `SQLFORM.factory()`.

```
form = SQLFORM.factory(
    Field('username', requires=IS_NOT_EMPTY()),
    Field('password', 'password', requires=IS_NOT_EMPTY()))
```

We need to create the fields for site and role with a drop-down menu. The `SQLFORM.factory()` provides the option using `IS_IN_SET()` function to display drop-down menu. The drop-down menu can list the content of Python’s data structure called set. First we will look at `SQLFORM.factory()`’s option for drop-down menu, and then we will spend some time in the Python interpreter to understand what is set.

```
form = SQLFORM.factory(
    Field('username', requires=IS_NOT_EMPTY()),
    Field('password', 'password', requires=IS_NOT_EMPTY()),
    Field('site', requires=IS_IN_SET(sites)),
    Field('role', requires=IS_IN_SET(roles)))
```

The “site” and “role” fields are going to show the drop-down menus with the list of sites and roles stored in the set “sites” and “roles” respectively. Now we will explore Python’s data structure set. We will create a list with duplicate entries, and then we will convert the list as set.

```
anees:~ anees$ python
>>>
>>> sites = ["All", "sjc", "atl", "lax", "sjc"]
>>>
>>> sites
['All', 'sjc', 'atl', 'lax', 'sjc']
>>>
>>> type(sites)
<type 'list'>
>>>
>>> sites = set(["All", "sjc", "atl", "lax", "sjc"])
>>>
>>> sites
set(['sjc', 'All', 'lax', 'atl'])
>>>
>>> type(sites)
<type 'set'>
```

Before defining the `SQLFORM.factory()`, we need to build the set `sites` and `roles` from the `inventory.json` file. We will read this file and pull site and role for each switch and add it to the set `sites` and `roles`. Since `sites` and `roles` are sets, we don't have to worry about the duplicate entries of site and role from the `inventory.json` file.

Let's start building this portion of the script step by step.

```
def packet_drops():
    # Read the inventory.json file into a dictionary
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define set sites and roles with a single item "All"
    sites = set(["All"])
    roles = set(["All"])

    # Read site and role of each switch and add it to the set
    for each_switch in inventory.keys():
        sites.add(inventory[each_switch]["site"])
        roles.add(inventory[each_switch]["role"])

    return dict(sites=sites, roles=roles)
```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/packet_drops.

Packet Drops

```
roles:set([u'spine', 'All', u'leaf'])
sites:set([u'sjc', 'All'])
```

Let's add the form and validate the display of the form.

```
def packet_drops():
    # Read the inventory.json file into a dictionary
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define set sites and roles with a single item "All"
```

```

sites = set(["All"])
roles = set(["All"])

# Read site and role of each switch and add it to the set
for each_switch in inventory.keys():
    sites.add(inventory[each_switch]["site"])
    roles.add(inventory[each_switch]["role"])

# Create a form using the sets
form = SQLFORM.factory(
    Field('username', requires=IS_NOT_EMPTY()),
    Field('password', 'password', requires=IS_NOT_EMPTY()),
    Field('site', requires=IS_IN_SET(sites)),
    Field('role', requires=IS_IN_SET(roles)),)

return dict(form=form)

```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/packet_drops.

Step 3: Build Switch IP Address list

Our goal is to derive the list of switch IP addresses from the inventory.json file based on the selection of site and role by the user. One important point to remember here is that we provide an option “All” in the site and role field of the form. So we need to write an algorithm to derive the switch IP addresses. Let us look at the inventory of the switches before writing the algorithm.

View Switches

```
inventory:172.28.170.114:hostname :22sw35
                        model      :DCS-7250QX-64
                        role        :spine
                        serialnumber:JPE14421537
                        site        :sjc
                        version     :4.15.3F
172.28.170.115:hostname :22sw37
                        model      :DCS-7250QX-64
                        role        :spine
                        serialnumber:JPE14402468
                        site        :sjc
                        version     :4.15.3F
172.28.170.97 :hostname :22sw2
                        model      :DCS-7050SX-128
                        role        :leaf
                        serialnumber:JPE14080457
                        site        :sjc
                        version     :4.15.3F
172.28.170.98 :hostname :22sw4
                        model      :DCS-7050SX-128
                        role        :leaf
                        serialnumber:JPE14080459
                        site        :sjc
                        version     :4.15.3F
```

Here is the algorithm we are going to use to build the list of switch IP addresses.

- Create an empty list called switches = []
- If the site = “All” and the role = “All”, we need all the switch IP addresses from the inventory.json file. So we will assign inventory.keys() to the list switches.
- If both the site and role are not “All”, we have to parse through site and role of each switch against the input selected by the user.
 - If the site = “All” and the role = (user selected), compare the role of each switch against the role selected by the user.
 - If the site = (user selected) and the role = “All” compare the site of each switch against the site selected by the user.
 - If the site = (user selected), role = (user selected), compare the site and role of each switch against the input selected by the user.

Let’s update the packet_drops() function to derive switch ip addresses after the user submitted the form.

```
def packet_drops():
    # Read the inventory.json file into a dictionary
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define set sites and roles with a single item "All"
    sites = set(["All"])
    roles = set(["All"])

    # Read site and role of each switch and add it to the set
    for each_switch in inventory.keys():
```

```

        sites.add(inventory[each_switch]["site"])
        roles.add(inventory[each_switch]["role"])

# Create a form using the sets
form = SQLFORM.factory(
    Field('username', requires=IS_NOT_EMPTY()),
    Field('password', 'password', requires=IS_NOT_EMPTY()),
    Field('site', requires=IS_IN_SET(sites)),
    Field('role', requires=IS_IN_SET(roles)),)

if form.process().accepted:
    # Create a switch IP addresses list from site and role selection
    switches = []

    """ compare the site and role between user selected and the values in
    inventory """

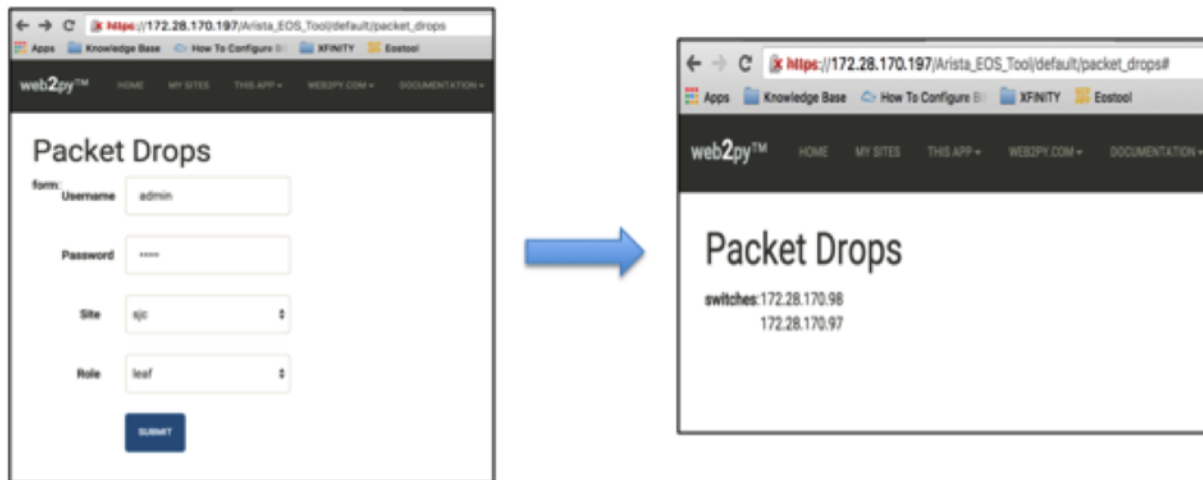
    if form.vars.site == "All" and form.vars.role == "All":
        switches = inventory.keys()
    else:
        for each_switch in inventory.keys():
            each_site = inventory[each_switch]["site"]
            each_role = inventory[each_switch]["role"]
            if form.vars.site == "All" and form.vars.role == each_role:
                switches.append(each_switch)
            elif (form.vars.site == each_site and
                  form.vars.role == "All"):
                switches.append(each_switch)
            elif (form.vars.site == each_site and
                  form.vars.role == each_role):
                switches.append(each_switch)

    # Return the switches list to the view for verification purpose
    return dict(switches=switches)

return dict(form=form)

```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/packet_drops. Make sure you see the expected switch IP addresses list by selecting various combinations of sites and roles.



Step 4: Reuse the packet drops Python script

We will use the packet drops Python script which we wrote in chapter 3. The only thing you should change in that script is to change the username and password variable in the `pyeapi.connect()` function. You should use `form.vars.username` and `form.vars.password` for username and password variable.

```
def packet_drops():
    # Read the inventory.json file into a dictionary
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define set sites and roles with a single item "All"
    sites = set(["All"])
    roles = set(["All"])

    # Read site and role of each switch and add it to the set
    for each_switch in inventory.keys():
        sites.add(inventory[each_switch]["site"])
        roles.add(inventory[each_switch]["role"])

    # Create a form using the sets
    form = SQLFORM.factory(
        Field('username', requires=IS_NOT_EMPTY()),
        Field('password', 'password', requires=IS_NOT_EMPTY()),
        Field('site', requires=IS_IN_SET(sites)),
        Field('role', requires=IS_IN_SET(roles)),
    )

    if form.process().accepted:
        # Create a switch IP addresses list from site and role selection
        switches = []

        """ compare the site and role between user selected and the values in
        inventory """

        if form.vars.site == "All" and form.vars.role == "All":
            switches = inventory.keys()
        else:
```

```

for each_switch in inventory.keys():
    each_site = inventory[each_switch]["site"]
    each_role = inventory[each_switch]["role"]
    if form.vars.site == "All" and form.vars.role == each_role:
        switches.append(each_switch)
    elif (form.vars.site == each_site and
          form.vars.role == "All"):
        switches.append(each_switch)
    elif (form.vars.site == each_site and
          form.vars.role == each_role):
        switches.append(each_switch)

# Packet Drops Script
interface_drops = {}
errors = {}
for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch,
                              username=form.vars.username,
                              password=form.vars.password,
                              port=None)

        # Execute the desired command
        interface_counters = node.execute(
            ["show interfaces counters discards"])
        interface_counters_clean = interface_counters[
            "result"][0]["interfaces"]
        host_name = node.execute(["show hostname"])
        host_name_clean = str(host_name["result"][0]["hostname"])
        interface_drops[host_name_clean] = {}

        for interface in interface_counters_clean.keys():
            if "Port-Channel" not in interface:
                if interface_counters_clean[interface]["inDiscards"] or
↪interface_counters_clean[interface]["outDiscards"] != 0:
                    interface_drops[host_name_clean][interface] = {}
                    show_interface = node.execute(
                        ["show interfaces " + str(interface)])
                    show_interface_clean = show_interface["result"][0][
                        "interfaces"][interface]["interfaceCounters"]
                    interface_drops[host_name_clean][interface]["Interface_
↪Status"] = show_interface[
                        "result"][0]["interfaces"][interface]["interfaceStatus
↪"]
                    interface_drops[host_name_clean][interface]["Line_
↪Protocol Status"] = show_interface[
                        "result"][0]["interfaces"][interface][
↪"lineProtocolStatus"]

                    if interface_counters_clean[interface]["inDiscards"] != 0:
                        interface_drops[host_name_clean][
                            interface]["inDiscards"] = {}
                        interface_drops[host_name_clean][interface][
↪"inDiscards"] [
                            "Total Discards"] = interface_counters_
↪clean[interface]["inDiscards"]
                        interface_drops[host_name_clean][interface][
↪"inDiscards"] [

```

```

                                "Input Errors"] = show_interface_clean[
↪ "inputErrorsDetail"]

                                if interface_counters_clean[interface]["outDiscards"] != 0:
↪ 0:
                                    interface_drops[host_name_clean][
                                        interface]["outDiscards"] = {}
                                    interface_drops[host_name_clean][interface][
↪ "outDiscards"] [
                                        "Total Discards"] = interface_counters_
↪ clean[interface] ["outDiscards"]
                                        interface_drops[host_name_clean][interface][
↪ "outDiscards"] [
                                        "Output Errors"] = show_interface_clean[
↪ "outputErrorsDetail"]

                                except pyeapi.eapilib.ConnectionError:
                                    errors[switch] = "ConnectionError: unable to connect to eAPI"

                                except pyeapi.eapilib.CommandError:
                                    errors[switch] = "CommandError: Check your EOS command syntax"

                                if not interface_drops[host_name_clean]:
                                    del interface_drops[host_name_clean]

                                # Return the switches list to the view for verification purpose
                                return dict(errors=errors, interface_drops=interface_drops)

                                return dict(form=form)

```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/packet_drops

Packet Drops		
interface_drops:22sw2	:Ethernet21:Interface Status	:connected
	Line Protocol Status:	up
	outDiscards	:Output Errors :{u'lateCollisions': 0, u'deferredTransmissions': 0, u'txPause': 0, u'collisions': 0}
		Total Discards:45941
	Ethernet95:Interface Status	:notconnect
	Line Protocol Status:	notPresent
	outDiscards	:Output Errors :{u'lateCollisions': 0, u'deferredTransmissions': 0, u'txPause': 0, u'collisions': 0}
		Total Discards:1
22sw35:Ethernet4/1	:Interface Status	:connected
	Line Protocol Status:	up
	outDiscards	:Output Errors :{u'lateCollisions': 0, u'deferredTransmissions': 0, u'txPause': 0, u'collisions': 0}
		Total Discards:1
22sw37:Ethernet1/1	:Interface Status	:connected
	Line Protocol Status:	up
	outDiscards	:Output Errors :{u'lateCollisions': 0, u'deferredTransmissions': 0, u'txPause': 0, u'collisions': 0}
		Total Discards:1222
	Ethernet3/1:Interface Status	:connected
	Line Protocol Status:	up
	outDiscards	:Output Errors :{u'lateCollisions': 0, u'deferredTransmissions': 0, u'txPause': 0, u'collisions': 0}
		Total Discards:5927
	Ethernet4/1:Interface Status	:connected
	Line Protocol Status:	up
	outDiscards	:Output Errors :{u'lateCollisions': 0, u'deferredTransmissions': 0, u'txPause': 0, u'collisions': 0}
		Total Discards:4716

Script Enhancements

At this point, we know how to port our scripts to web2py. To summarize, we have three sections in the script.

1. Web2Py form to receive user input.
2. Build Switch List based on the user input.
3. The core logic for your Network Use case.

The first two sections are going to be common for the most use cases. We can create functions for these two sections so that we can re-use them for all our use cases. We will also create a function for discovering the packet drops which will improve the readability and functionality of the script.

Step 1: Create a function to create a web2py form

We will create a function called `eos_form` to build a web2py form. We will call this form from the `packet_drops` function.

```
def eos_form():
    # Read the inventory.json file into a dictionary
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define set sites and roles with a single item "All"
    sites = set(["All"])
    roles = set(["All"])

    # Read site and role of each switch and add it to the set
    for each_switch in inventory.keys():
        sites.add(inventory[each_switch]["site"])
        roles.add(inventory[each_switch]["role"])

    # Create a form using the sets
    form = SQLFORM.factory(
        Field('username', requires=IS_NOT_EMPTY()),
        Field('password', 'password', requires=IS_NOT_EMPTY()),
        Field('site', requires=IS_IN_SET(sites)),
        Field('role', requires=IS_IN_SET(roles)),)

    return form

def packet_drops():
    # Build Form
    form = eos_form()

    if form.process().accepted:
        # Create a switch IP addresses list from site and role selection
        switches = []

        """ compare the site and role between user selected and the values in
        inventory """
```

Step 2: Create a function to derive the list of switches

We will create a function `switches_list` to derive the list of switch IP addresses. We will call this function from the `packet_drops` function. When we call, we need to pass the user input so that `switches_list` function derives the list from the user provides values for site and role.

```
def switches_list(form_vars):
    # Read the inventory
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define an empty list
    switches = []

    """ compare the site and role between user selected and the values in
        inventory """
    if form_vars.site == "All" and form_vars.role == "All":
        switches = inventory.keys()
    else:
        for each_switch in inventory.keys():
            each_site = inventory[each_switch]["site"]
            each_role = inventory[each_switch]["role"]
            if form_vars.site == "All" and form_vars.role == each_role:
                switches.append(each_switch)
            elif form_vars.site == each_site and form_vars.role == "All":
                switches.append(each_switch)
            elif form_vars.site == each_site and form_vars.role == each_role:
                switches.append(each_switch)

    return switches

def packet_drops():
    # Build Form
    form = eos_form()

    if form.process().accepted:
        # Create a switch IP addresses list from site and role selection
        switches = switches_list(form.vars)

        # Packet Drops Script
        interface_drops = {}
        errors = {}
        for switch in switches:
```

Step 3: Create a function for discovering interface drops

We will write a function for discovering interface drops and we will call this function from packet_drops function.

```
def discover_packet_drops(node):
    # Define empty dictionary
    int_drops = {}

    # Execute the desired command
    int_counters_raw = node.execute(["show interfaces counters discards"])
    int_counters = int_counters_raw["result"][0]["interfaces"]

    for interface in int_counters.keys():
        if "Port-Channel" not in interface:
            if (int_counters[interface]["inDiscards"] or
                int_counters[interface]["outDiscards"] != 0):
                int_drops[interface] = {}
                show_int_raw = node.execute(
```

```

        ["show interfaces " + str(interface)])
    show_int = show_int_raw["result"][0][
        "interfaces"][interface]["interfaceCounters"]
    int_drops[interface]["Interface Status"] = show_int_raw[
        "result"][0]["interfaces"][interface]["interfaceStatus"]
    int_drops[interface]["Line Protocol Status"] = show_int_raw[
        "result"][0]["interfaces"][interface]["lineProtocolStatus"]

    if int_counters[interface]["inDiscards"] != 0:
        int_drops[interface]["inDiscards"] = {}
        int_drops[interface]["inDiscards"][
            "Total Discards"] = int_counters[interface]["inDiscards"]
        int_drops[interface]["inDiscards"][
            "Input Errors"] = show_int["inputErrorsDetail"]

    if int_counters[interface]["outDiscards"] != 0:
        int_drops[interface]["outDiscards"] = {}
        int_drops[interface]["outDiscards"][
            "Total Discards"] = int_counters[interface]["outDiscards"]
        int_drops[interface]["outDiscards"][
            "Output Errors"] = show_int["outputErrorsDetail"]

    return int_drops

def host_name(node):
    host_name = node.execute(["show hostname"])
    host_name_clean = str(host_name["result"][0]["hostname"])
    return host_name_clean

def packet_drops():
    # Build Form
    form = eos_form()

    if form.process().accepted:
        # Create a switch IP addresses list from site and role selection
        switches = switches_list(form.vars)

        # Packet Drops Script
        interface_drops = {}
        errors = {}
        for switch in switches:
            try:
                # Define API Connection String
                node = pyeapi.connect(transport="https", host=switch,
                                     username=form.vars.username,
                                     password=form.vars.password,
                                     port=None)

                # Identify the switch hostname for reporting purpose
                switchname = host_name(node)

                # Discover Interface Packet Drops
                interface_drops[switchname] = discover_packet_drops(node)

            except pyeapi.eapilib.ConnectionError:
                errors[switch] = "ConnectionError: unable to connect to eAPI"

```

```
    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

    if not interface_drops[switchname]:
        del interface_drops[switchname]

    # Return the switches list to the view for verification purpose
    return dict(errors=errors, interface_drops=interface_drops)

return dict(form=form)
```

Final Script

The final script with all the functions is shown below.

```
def eos_form():
    # Read the inventory.json file into a dictionary
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define set sites and roles with a single item "All"
    sites = set(["All"])
    roles = set(["All"])

    # Read site and role of each switch and add it to the set
    for each_switch in inventory.keys():
        sites.add(inventory[each_switch]["site"])
        roles.add(inventory[each_switch]["role"])

    # Create a form using the sets
    form = SQLFORM.factory(
        Field('username', requires=IS_NOT_EMPTY()),
        Field('password', 'password', requires=IS_NOT_EMPTY()),
        Field('site', requires=IS_IN_SET(sites)),
        Field('role', requires=IS_IN_SET(roles)),)

    return form

def switches_list(form_vars):
    # Read the inventory
    with open(file) as readfile:
        inventory = json.load(readfile)

    # Define an empty list
    switches = []

    """ compare the site and role between user selected and the values in
        inventory """
    if form_vars.site == "All" and form_vars.role == "All":
        switches = inventory.keys()
    else:
        for each_switch in inventory.keys():
            each_site = inventory[each_switch]["site"]
            each_role = inventory[each_switch]["role"]
            if form_vars.site == "All" and form_vars.role == each_role:
```

```

        switches.append(each_switch)
    elif form_vars.site == each_site and form_vars.role == "All":
        switches.append(each_switch)
    elif form_vars.site == each_site and form_vars.role == each_role:
        switches.append(each_switch)

    return switches

def discover_packet_drops(node):
    # Define empty dictionary
    int_drops = {}

    # Execute the desired command
    int_counters_raw = node.execute(["show interfaces counters discards"])
    int_counters = int_counters_raw["result"][0]["interfaces"]

    for interface in int_counters.keys():
        if "Port-Channel" not in interface:
            if (int_counters[interface]["inDiscards"] or
                int_counters[interface]["outDiscards"] != 0):
                int_drops[interface] = {}
                show_int_raw = node.execute(
                    ["show interfaces " + str(interface)])
                show_int = show_int_raw["result"][0][
                    "interfaces"][interface]["interfaceCounters"]
                int_drops[interface]["Interface Status"] = show_int_raw[
                    "result"][0]["interfaces"][interface]["interfaceStatus"]
                int_drops[interface]["Line Protocol Status"] = show_int_raw[
                    "result"][0]["interfaces"][interface]["lineProtocolStatus"]

                if int_counters[interface]["inDiscards"] != 0:
                    int_drops[interface]["inDiscards"] = {}
                    int_drops[interface]["inDiscards"][
                        "Total Discards"] = int_counters[interface]["inDiscards"]
                    int_drops[interface]["inDiscards"][
                        "Input Errors"] = show_int["inputErrorsDetail"]

                if int_counters[interface]["outDiscards"] != 0:
                    int_drops[interface]["outDiscards"] = {}
                    int_drops[interface]["outDiscards"][
                        "Total Discards"] = int_counters[interface]["outDiscards"]
                    int_drops[interface]["outDiscards"][
                        "Output Errors"] = show_int["outputErrorsDetail"]

    return int_drops

def host_name(node):
    host_name = node.execute(["show hostname"])
    host_name_clean = str(host_name["result"][0]["hostname"])
    return host_name_clean

def packet_drops():
    # Build Form
    form = eos_form()

    if form.process().accepted:

```

```
# Create a switch IP addresses list from site and role selection
switches = switches_list(form.vars)

# Packet Drops Script
interface_drops = {}
errors = {}
for switch in switches:
    try:
        # Define API Connection String
        node = pyeapi.connect(transport="https", host=switch,
                               username=form.vars.username,
                               password=form.vars.password,
                               port=None)

        # Identify the switch hostname for reporting purpose
        switchname = host_name(node)

        # Discover Interface Packet Drops
        interface_drops[switchname] = discover_packet_drops(node)

    except pyeapi.eapilib.ConnectionError:
        errors[switch] = "ConnectionError: unable to connect to eAPI"

    except pyeapi.eapilib.CommandError:
        errors[switch] = "CommandError: Check your EOS command syntax"

    if not interface_drops[switchname]:
        del interface_drops[switchname]

# Return the switches list to the view for verification purpose
return dict(errors=errors, interface_drops=interface_drops)

return dict(form=form)
```

Control Plane Drops

This is going to be a very simple script. We have already written functions for form and switch list. We have also written script for control plane drops in the chapter 3.

We will write a new function called `discover_copp_drops()` in `default.py` and re-use the script for the control plane drops that we wrote in the chapter 3. Then we will create another function called `controlplane_drops()` in `default.py` to build our use-case by using all the three functions.

```
def discover_copp_drops(node):
    # Define empty dictionary
    copp_drops = {}

    # Execute the desired command
    copp_counters_raw = node.execute(
        ["show policy-map interface control-plane copp-system-policy"])
    copp_counters = copp_counters_raw["result"][0][
        "policyMaps"]["copp-system-policy"]["classMaps"]

    # parse through each copp system class map and discover non drop counters
    for each_copp_class in copp_counters.keys():
        if (copp_counters[each_copp_class]["intfPacketCounters"]
```

```

        ["dropPackets"] != 0):
            copp_drops[each_copp_class] = {}
            copp_drops[each_copp_class]["Drop Packets"] = copp_counters[
                each_copp_class]["intfPacketCounters"]["dropPackets"]

    return copp_drops

def controlplane_drops():
    # Build Form
    form = eos_form()

    if form.process().accepted:
        # Create a switch IP addresses list from site and role selection
        switches = switches_list(form.vars)

        # CoPP Drops Script
        copp_drops = {}
        errors = {}
        for switch in switches:
            try:
                # Define API Connection String
                node = pyeapi.connect(transport="https", host=switch,
                                     username=form.vars.username,
                                     password=form.vars.password,
                                     port=None)

                # Identify the switch hostname for reporting purpose
                switchname = host_name(node)

                # Discover CoPP Drops
                copp_drops[switchname] = discover_copp_drops(node)

            except pyeapi.eapilib.ConnectionError:
                errors[switch] = "ConnectionError: unable to connect to eAPI"

            except pyeapi.eapilib.CommandError:
                errors[switch] = "CommandError: Check your EOS command syntax"

            if not copp_drops[switchname]:
                del copp_drops[switchname]

        # Return the switches list to the view for verification purpose
        return dict(errors=errors, copp_drops=copp_drops)

    return dict(form=form)

```

Create a View for the function `controlplane_drops`

Click the “files toggle” on the top left

Click Create and select views from the drop down window

Provide the file name with path → default/controlplane_drops.html

We are going to create a default view with the header “Control Plane Drops”. Save this file.

```

{{extend 'layout.html'}}
<h1>Control Plane Drops</h1>

```

```
{%=BEAUTIFY(response._vars)%}
```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/controlplane_drops

Control Plane Drops

```
copp_drops:22sw2 :copp-system-glean:Drop Packets:26878256
            22sw37:copp-system-default:Drop Packets:1225931
            copp-system-glean :Drop Packets:69998
            22sw4 :copp-system-glean:Drop Packets:29006360
errors      :
```

Last 10 Sev 1-3 Log Messages

The goal of this script is to collect the last 10 severity 1-3 messages from the switches. It will be helpful when you are troubleshooting a network-wide problem and quickly run this command and look at the severity 1-3 logs on the switches within the network. We have not written a script for this logic previously in this book. So we will first explore the logic through IDLE and then we will port the logic into this tool.

Let's login to an Arista switch and explore the required EOS commands.

```
Switch# show logging | json
% This is an unconverted command

switch# show logging 10 errors

Jan  9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.
↪10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes

Jan  9 10:55:57 22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.
↪10.211.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
```

Before looking at the algorithm, let's take a look at the basic Python script to pull the logs from the switch. Since the show log is not json converted, we will use the jsonrpclib with "text" format, and we parse the data using the string parsing methods.

```
import pprint
from jsonrpclib import Server
import ssl
ssl._create_default_https_context = ssl._create_unverified_context

node = Server("https://admin:admin@172.28.170.97/command-api")

show_log = node.runCmds(1, ["show logging 10 errors"], "text")
show_log_clean = show_log[0]["output"]

pprint.pprint(show_log_clean)
```

Save and run this script.


```
>>> ===== RESTART =====
>>>
u'Jan  9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.
↪10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes\nJan  9
↪10:55:57 22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.211.
↪11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes\n'
```

Algorithm

We will build the script directly from web2py editor. We are going to write a function `switch_logs()` in the default controller of our web2py application `Arista_EOS_tool`. We are going to use the below algorithm to write the script.

1. Create a New Function and a View for this task “show logs.”
2. Use the `eos_form()` and `switches_list()` function to display form.
3. Write the core logic of the script
 1. Create a list for the “logging” levels 0 to 3, which are emergencies, alerts, critical and errors.
 2. For each “logging” level within the list, collect the show logging 10 <logging_level> output.
 3. For each logging level, parse through the required output and store under the corresponding switch entry within the dictionary.
4. Return the dictionary `show_log` to the view.

Develop Script

Step 1: Create a New Function and a View for this task “show logs”

Create a new function `switch_logs()` in the default.py controller.

```
def switch_logs():
    return dict()
```

Create a view `switch_logs.html` under `views/default` folder.

```
{{extend 'layout.html'}}
<h1>Display Logs with Severity 0 to 3 </h1>
{{=BEAUTIFY(response._vars)}}
```

Step 2: Use the `eos_form()` and `switches_list()` function to display form

We have written two functions `eos_form()` and `switches_list()` in the previous use case. We will use those functions with this script to display form and derive IP addresses of the switches from the user selection.

```
def switch_logs():
    # Build Form
    form = eos_form()

    if form.process().accepted:
        # Create a switch IP addresses list from site and role selection
        switches = switches_list(form.vars)
```

```
return dict(form=form)
```

Step 3: Write the core logic of the script

Since we are going to use jsonrpclib, we need to install that module in the Linux system where we are hosting the web2py application.

```
anees@ubuntu-web2py:~$ sudo pip2 install jsonrpclib
Downloading/unpacking jsonrpclib
  Downloading jsonrpclib-0.1.7.tar.gz
  Running setup.py (path:/tmp/pip_build_root/jsonrpclib/setup.py) egg_info for
↳ package jsonrpclib

Installing collected packages: jsonrpclib
  Running setup.py install for jsonrpclib

Successfully installed jsonrpclib
Cleaning up...
```

The algorithm for the core logic is shown below.

1. Create a list for the “logging” levels 0 to 3, which are emergencies, alerts, critical and errors.
2. For each “logging” level within the list, collect the show logging 10 <logging_level> output.
3. For each logging level, parse through the required output and store under the corresponding switch entry within the dictionary.

```
# Import the neccessary python modules
import pyeapi
import json
from jsonrpclib import Server, ProtocolError

def hostname(node):
    host_name = node.runCmds(1, ["show hostname"])
    host_name_clean = str(host_name[0]["hostname"])
    return host_name_clean

def collect_logs(node):
    # Define an empty list
    show_log = []

    # Create a list for the logging levels 0 to 3
    severity_level = ["emergencies", "alerts", "critical", "errors"]

    # For each logging level listed in the list, collect the show log
    for each_sev_level in severity_level:
        show_log_cli_raw = node.runCmds(
            1, ["show logging 10 " + each_sev_level], "text")
        show_log_cli = show_log_cli_raw[0]["output"]
        show_log.append(show_log_cli)

    return show_log
```

```
def switch_logs():
    form = eos_form()
    if form.process().accepted:
        form_vars = form.vars
        switches = switches_list(form_vars)

        # Define dictionaries
        show_log = {}

        for switch in switches:
            node = Server("https://" + form_vars.username + ":" +
                          form_vars.password + "@" + switch +
                          "/command-api")

            # Collect hostname for reporting purpose
            name = hostname(node)

            # Collect the logs
            show_log[name] = collect_logs(node)

        return dict(show_log=show_log)

    return dict(form=form)
```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/switch_logs.

Display Logs with Severity 0 to 3

```
show_log:22sw2 :Jan 9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes Jan 9 10:55:57
22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.211.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes Jan 9 10:55:57
22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.211.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes Jan 9 10:55:57
22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.211.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
22sw35:
22sw37:
mn1 :
```

We are going to add three things to complete this script

1. Add Exception Handling
2. Display log entry one per line for clean view of the output
3. Add a logic to delete the switch entries if there are no logs.

```
def collect_logs(node):
    # Define an empty list
    show_log = []

    # Create a list for the logging levels 0 to 3
    severity_level = ["emergencies", "alerts", "critical", "errors"]

    # For each logging level listed in the list, collect the show log
    for each_sev_level in severity_level:
        show_log_cli_raw = node.runCmds(
            1, ["show logging 10 "+each_sev_level], "text")
        show_log_cli = show_log_cli_raw[0]["output"]
        for line in show_log_cli.splitlines():
```

```
        show_log.append(line)

    return show_log

def switch_logs():
    form = eos_form()
    if form.process().accepted:
        form_vars = form.vars
        switches = switches_list(form_vars)

        # Define dictionaries
        show_log = {}
        errors = {}

        for switch in switches:
            try:
                node = Server("https://" + form_vars.username + ":" +
                              form_vars.password + "@" + switch +
                              "/command-api")

                # Collect hostname for reporting purpose
                name = hostname(node)

                # Collect the logs
                show_log[name] = collect_logs(node)

                # If there are no logs, delete the entry for the switch
                if not show_log[name]:
                    del show_log[name]

            except ProtocolError as e:
                errors[switch] = "Invalid EOS Command" + str(e)

            except:
                errors[switch] = "eAPI Connection Error"

        return dict(errors=errors, show_log=show_log)

    return dict(form=form)
```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/switch_logs.

Display Logs with Severity 0 to 3

```
errors :
show_log:22sw2:Jan 9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.211.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.211.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.211.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-102[27895]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.202.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
Jan 9 10:55:57 22sw2 ribd-vrf-111[28249]: %BGP-3-NOTIFICATION: sent to neighbor 10.10.211.11 (AS 99) 4/0 (Hold Timer Expired Error/Unspecified) 0 bytes
```

Final Script

The final script with all the functions is shown below.

```

# Import the necessary python modules
from jsonrpclib import Server, ProtocolError

def hostname(node):
    host_name = node.runCmds(1, ["show hostname"])
    host_name_clean = str(host_name[0]["hostname"])
    return host_name_clean

def collect_logs(node):
    # Define an empty list
    show_log = []

    # Create a list for the logging levels 0 to 3
    severity_level = ["emergencies", "alerts", "critical", "errors"]

    # For each logging level listed in the list, collect the show log
    for each_sev_level in severity_level:
        show_log_cli_raw = node.runCmds(
            1, ["show logging 10 "+each_sev_level], "text")
        show_log_cli = show_log_cli_raw[0]["output"]
        for line in show_log_cli.splitlines():
            show_log.append(line)

    return show_log

def switch_logs():
    form = eos_form()
    if form.process().accepted:
        form_vars = form.vars
        switches = switches_list(form_vars)

        # Define dictionaries
        show_log = {}
        errors = {}

        for switch in switches:
            try:
                node = Server("https://" + form_vars.username + ":" +
                              form_vars.password + "@" + switch +
                              "/command-api")

                # Collect hostname for reporting purpose
                name = hostname(node)

                # Collect the logs
                show_log[name] = collect_logs(node)

                # If there are no logs, delete the entry for the switch
                if not show_log[name]:
                    del show_log[name]

            except ProtocolError as e:
                errors[switch] = "Invalid EOS Command" + str(e)

        except:

```

```
        errors[switch] = "eAPI Connection Error"

    return dict(errors=errors, show_log=show_log)

return dict(form=form)
```

Chapter 10: Web2Py - Capacity Planning Use Cases

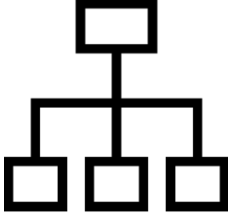
- *Port Capacity*
 - *Develop Script*
 - * *Step 1: Create a new Function and a View for this task “Port Capacity”*
 - * *Step 2: Use the eos_form() and switches_list() function to display form*
 - * *Step 3: Create a function to discover the unused ports*
- *Hardware Scale*
- *Summary*

We have created a solid framework using web2py in the previous two sections “Prepare the Tool” and “Troubleshooting Use Cases.” This section should be pretty straightforward to create the use cases for Capacity Planning with the framework. And that’s the goal of this book. Once we created the framework, you should be able to add plenty of use cases by spending time only on the core logic of your requirement than spending time on the web2py framework.

In this chapter, we are going to write two use cases that are Port Capacity and Hardware Tables Capacity.


Arista [HOME](#) [ARISTA LINKS](#) [OPERATIONS RUN BOOK](#) [LOG IN](#)

Arista Professional Services




Prepare the Tool

1. Add Switches
2. Categorize Switches
3. View Switches



Capacity Planning

1. Port Capacity
2. Hardware Tables Capacity (EMT, LPM, and TCAM)



Network-wide Troubleshooting

1. Packet Drops
2. Control Plane Issues (T2 Only)
3. Last 10 Sev 1-3 Log messages

Copyright © 2016 Powered by [web2py](#)

Port Capacity

The goal of this use case is to find the unused ports and transceivers in the network. We have already created the Python script for this use case earlier in this book. From web2py perspective, when the user accesses the Port Capacity link from the home page, it should show the same form that asks for username, password, site and role.

form : Username:

Password:

Network:

✓ sjc

Role:

All

Once the user provided the required information, the script collects the information about the unused ports and displays the result.


```

unused_ports :
  22sw2 :
    10 :
      10GBASE-SRL : 1
      Not Present : 92
      dot1q-encapsulation : 3
    40 :
      Not Present : 5
    description : 96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU
    model : DCS-7050SX-128
    serialnumber : JPE14080457
  22sw35 :
    10 :
      Not Present : 216
    40 :
      Not Present : 1
    description : 64x QSFP+ 2RU
    model : DCS-7250QX-64
    serialnumber : JPE14421537

```

We are going to use the following steps to write the script.

Develop Script

1. Create a new Function and a View for this task “Port Capacity.”
2. Use the eos_form() and switches_list() function to display form.
3. Create a function to discover the unused ports.

Step 1: Create a new Function and a View for this task “Port Capacity”

Create a new function port_capacity() in the default.py controller.

```
def port_capacity():
    return dict()
```

Create a view port_capacity.html under views/default folder.

```
{{extend 'layout.html'}}
<h1>Port Capacity </h1>
{{=BEAUTIFY(response._vars)}}
```

Step 2: Use the eos_form() and switches_list() function to display form

We have written two functions eos_form() and switches_list() in the previous chapter. We will use those functions with this script to display form and derive IP addresses of the switches from the user selection.

```
def port_capacity():
    # Build Form
    form = eos_form()
```

```
if form.process().accepted:
    # Create a switch IP addresses list from site and role selection
    switches = switches_list(form.vars)

return dict(form=form)
```

Step 3: Create a function to discover the unused ports

We will create a new function `discover_unused_ports()` and reuse the script that we wrote in chapter 4. Then we will call this function from the `port_capacity()` function.

```
def discover_unused_ports(node):
    # Define a dictionary for unused_ports
    unused_ports = {}

    # Collect the model name and device description
    show_inventory = node.execute(["show inventory"])
    model = str(show_inventory["result"][0]["systemInformation"]["name"])
    description = str(show_inventory["result"][0]["systemInformation"]["description"])
    unused_ports = {"Model": model, "Description": description}

    # Collect interfaces status
    sh_int_status_raw = node.execute(["show interfaces status notconnect disabled"])
    sh_int_status = sh_int_status_raw["result"][0]["interfaceStatuses"]

    for each_interface in sh_int_status.keys():
        bandwidth = sh_int_status[each_interface]["bandwidth"]
        bandwidth_GE = str(bandwidth / 1000000000) + "GE"
        interface_type = str(sh_int_status[each_interface]["interfaceType"])

        # check for bandwidth entry and add it if not there
        if bandwidth_GE not in unused_ports:
            unused_ports[bandwidth_GE] = {}

        # check for interface type and add it if not there
        if interface_type not in unused_ports[bandwidth_GE]:
            unused_ports[bandwidth_GE][interface_type] = 1
        else:
            unused_ports[bandwidth_GE][interface_type] += 1

    return unused_ports

def port_capacity():
    form = eos_form()
    if form.process().accepted:
        switches = switches_list(form.vars)

        # Create an Empty Dictionary
        unused_ports = {}
        errors = {}

        for switch in switches:
```

```
try:
    # Define API Connection String
    node = pyeapi.connect(transport="https", host=switch,
                          username=form_vars.username,
                          password=form_vars.password,
                          port=None)

    # Collect hostname for reporting purpose
    switchname = host_name(node)

    # Discover Unused Ports
    unused_ports[switchname] = discover_unused_ports(node)

    # If there are no unused ports, delete the entry for the switch
    if not unused_ports[switchname]:
        del unused_ports[switchname]

except pyeapi.eapilib.ConnectionError:
    errors[switch] = "ConnectionError: unable to connect to eAPI"

except pyeapi.eapilib.CommandError:
    errors[switch] = "CommandError: Check your EOS command syntax"

return dict(errors=errors, unused_ports=unused_ports)

return dict(form=form)
```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/port_capacity.

Port Capacity

```
errors      :
unused_ports:22sw2 :0GE      :10/100/1000:1
                  N/A      :1
                  10GE     :10GBASE-SRL :1
                  Not Present :92
                  dot1q-encapsulation:3
                  40GE     :Not Present:5
                  Description:96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU
                  Model    :DCS-7050SX-128
22sw35:0GE     :10/100/1000:1
            10GE     :Not Present:216
            40GE     :Not Present:1
            Description:64x QSFP+ 2RU
            Model    :DCS-7250QX-64
22sw37:0GE     :10/100/1000:1
            N/A      :1
            10GE     :40GBASE-CR4:3
            Not Present :220
            Description:64x QSFP+ 2RU
            Model    :DCS-7250QX-64
mn1 :0GE      :10/100/1000:1
     10GE     :10GBASE-SRL:1
           Not Present :93
     40GE     :Not Present:5
     Description:96x 10Gb SFP+ + 8x 40Gb QSFP+ 2RU
     Model    :DCS-7050SX-128
```

Hardware Scale

We have built four use cases in the web2py. By now, you should be comfortable to build use cases in web2py by leveraging the existing functions and Python scripts. In this case, we are going to copy all the functions we created for hardware table scalability in chapter 4 in the web2py's default.py controller.

```
def mac_scale(node):
    show_mac = node.runCmds(1, ["show mac address-table count"])
    show_mac_clean = show_mac[0]["vlanCounts"]
    mac_count = 0
    for each_vlan in show_mac_clean.keys():
        mac_count += show_mac_clean[each_vlan]["dynamic"]
    return mac_count

def vrf_scale(node):
    show_vrf = node.runCmds(1, ["show vrf | include ipv4,ipv6"], "text")
    show_vrf_clean = show_vrf[0]["output"]
    vrfs = []
    for line in show_vrf_clean.splitlines():
        fields = line.split()
        vrfs.append(fields[0])
    return vrfs
```

```

def arp_scale(node, vrfs):
    arp_count = 0
    for each_vrf in vrfs:
        show_arp = node.runCmds(1, [
            "show ip arp vrf " + each_vrf + " summary"])
        arp_count += show_arp[0]["dynamicEntries"]
    return arp_count

def route_scale(node, vrfs):
    route_count = 0
    for each_vrf in vrfs:
        show_route = node.runCmds(1, [
            "show ip route vrf " + each_vrf + " summary"])
        route_count += show_route[0]["totalRoutes"]
    return route_count

def tcam_scale(node):
    tcam_count = 0
    show_tcam = node.runCmds(1, [
        "enable", "show platform trident tcam | include TCAM group"], "text")
    for each_line in show_tcam[1]["output"].splitlines():
        tcam_count += int(each_line.split()[4])
    return tcam_count

def hostname(node):
    host_name = node.runCmds(1, ["show hostname"])
    host_name_clean = str(host_name[0]["hostname"])
    return host_name_clean

def hardware_scale():
    form = eos_form()
    if form.process().accepted:
        form_vars = form.vars
        switches = switches_list(form_vars)

        # Hardware scale assessment script
        verify_scale = {}

        for switch in switches:
            try:
                # Define Connection Attributes for jsonrpc
                node = Server("https://" + form_vars.username + ":" +
                    form_vars.password + "@" + switch +
                    "/command-api")

                # Call the hardware scale functions
                name = hostname(node)
                mac_count = mac_scale(node)
                vrfs = vrf_scale(node)
                arp_count = arp_scale(node, vrfs)
                route_count = route_scale(node, vrfs)
                tcam_count = tcam_scale(node)

```

```
# Store the values in a dictionary

verify_scale[name] = {"MAC Scale": mac_count,
                     "Number of VRFs": len(vrfs),
                     "Number of ARP Entries": arp_count,
                     "Number of Routes": route_count,
                     "Number of TCAM entries Used": tcam_count
                     }

except ProtocolError as e:
    verify_scale[switch] = "Invalid EOS Command" + str(e)

except:
    verify_scale[switch] = "eAPI Connection Error"

return dict(verify_scale=verify_scale)

return dict(form=form)
```

Test your script using the URL https://<web-server>/Arista_EOS_Tool/default/hardware_scale.

Hardware Scale

```
verify_scale:22sw2 :MAC Scale           :5
                  Number of ARP Entries :33
                  Number of Routes      :234
                  Number of TCAM entries used:98
                  Number of VRFs        :16
22sw35:MAC Scale           :0
                  Number of ARP Entries :2050
                  Number of Routes      :6307
                  Number of TCAM entries used:510
                  Number of VRFs        :17
22sw37:MAC Scale           :0
                  Number of ARP Entries :2046
                  Number of Routes      :6273
                  Number of TCAM entries used:510
                  Number of VRFs        :18
mn1 :MAC Scale           :0
                  Number of ARP Entries :0
                  Number of Routes      :0
                  Number of TCAM entries used:96
                  Number of VRFs        :0
```

Summary

We have ported all the five use cases to the tool. You make a list of your common networking tasks and write the scripts using Python and then port it to the tool. You can also keep all the network-related documentation in this tool. Let's go to the next chapter to learn about the web2py view.