# Development of Python API for a Network Switch

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology
in

**Computer Science and Engineering**

By

**Archit Singh (151276)**

Under the supervision of

**Mr. Rangarajan Ramalingam**

to

Department of Computer Science & Engineering and Information Technology
**Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh**

# Acknowledgements

I express my profound gratitude and indebtedness to of **Mr Rangarajan Ramalinagm** Lead Architect Wipro limited for introducing the present topic and for their inspiring intellectual guidance, constructive criticism and valuable suggestion throughout this journey.

I am also thankful to all **Mr Vijay Natarajan** Competency Manager Wipro Limited for his constant motivation and helping me bring in improvements in the project.

Finally I would like to thank my family and friends for their constant support. Without their contribution it would have been impossible to complete my work.

**Date**
**Archit Singh**

# Table of Contents

## LIST OF FIGURES

## Abstract

The Cisco Nexus 1000v Switch is a software developed by Cisco in collaboration with VMware, Citrix and Redhat for VMware environments, Which runs inside VMware hypervisor (ESX), and supports Cisco server virtualization technology. It has two Components, The Virtual Module (VEM) and Virtual Supervisor Module (VSM). VEM executes commands inside the hypervisor and VSM manages the VEMs, as there are plenty of VEMs inside the Switch.

The existing CURL commands allow user to work with APIs but user actually has to get into the switches and execute the commands but by using python APIs we can automate the switches and use the methods accordingly.

In this project we were working on building REST APIs using Python language for accessing our network switch and managing Port Profiles, CLIs, VxLANs etc. Designing an API that has a built-in authentication mechanism that uses authentication calls to create a secure session and prevent unauthorized users from gaining access. An API that adheres to the principles of REST does not require the client to know anything about the structure of API. Rather the server needs to provide whatever information the client needs to interact with the service.

All the documentations and citations were referred from the official site of Cisco and all the related libraries were downloaded and used accordingly.

# Chapter 1. Introduction

## 1.1 Introduction

Application Programming Interface(API) is an intermediate that enables two applications to converse with one another. In essential terms, it goes about as a separation that passes on a requesting to the provider that is referenced and starting their response is delivered back. For example, Google or Facebook auto login features in many of the websites, and applications like Goibibo or MakeMyTrip, which uses same concept, getting details from different flights and showing it in a sorted way and then redirecting to that same flight where you want to book your seats.

**Representational State Transfer** (**REST**).It is an Application Program Interface (API) which uses HTTP requests, and also XML and JSON formats to Read (GET),Write (POST), Update(PUT) and Remove (DELETE) any data from the server side. There are so many methods to use an API like SOAP (Simple Object Access Protocol) and REST. But REST is more favorable as it uses less bandwidth, which makes it more suitable for Interanet.

In this project we have developed Python APIs for the Cisco Nexus 1000v Switch, which provides,

● Policy-based virtual machine (VM) connectivity.

● Mobile VM security and network policy.

● Non-disruptive operational model for your server virtualization, and networking teams.

It is fully integrated with VMware Virtual Infrastructure. We can use the Cisco Nexus 1000v to manage their VM connectivity and integrate the server virtualization infrastructure. Since virtual servers are not managed the same way as any physical servers, Server virtualization is treated as a special deployment, with longer deployment

time and great connectivity over network, among servers, storage and administrators. But in Cisco Nexus 1000v switch we can have a predictable systems administration include set and provisioning process right from the VM to the access, accumulation, and center switches. Your virtual servers can utilize a similar network configuration, tools, operational models security arrangement policies as physical servers.
Virtualization Administrators can use predefined arrange approach of network policy that pursues the Virtual Machine and spotlights on VM organization. This extensive arrangement of capacities encourages you to send server virtualization quicker and understand its advantages sooner.

The Cisco Nexus 1000v Switch has two major components, named as the Virtual Ethernet Module (VEM) and Virtual Supervisor Module (VSM). VEM executes inside the hypervisor and on the other hand VSM manages the VEMs.[1]

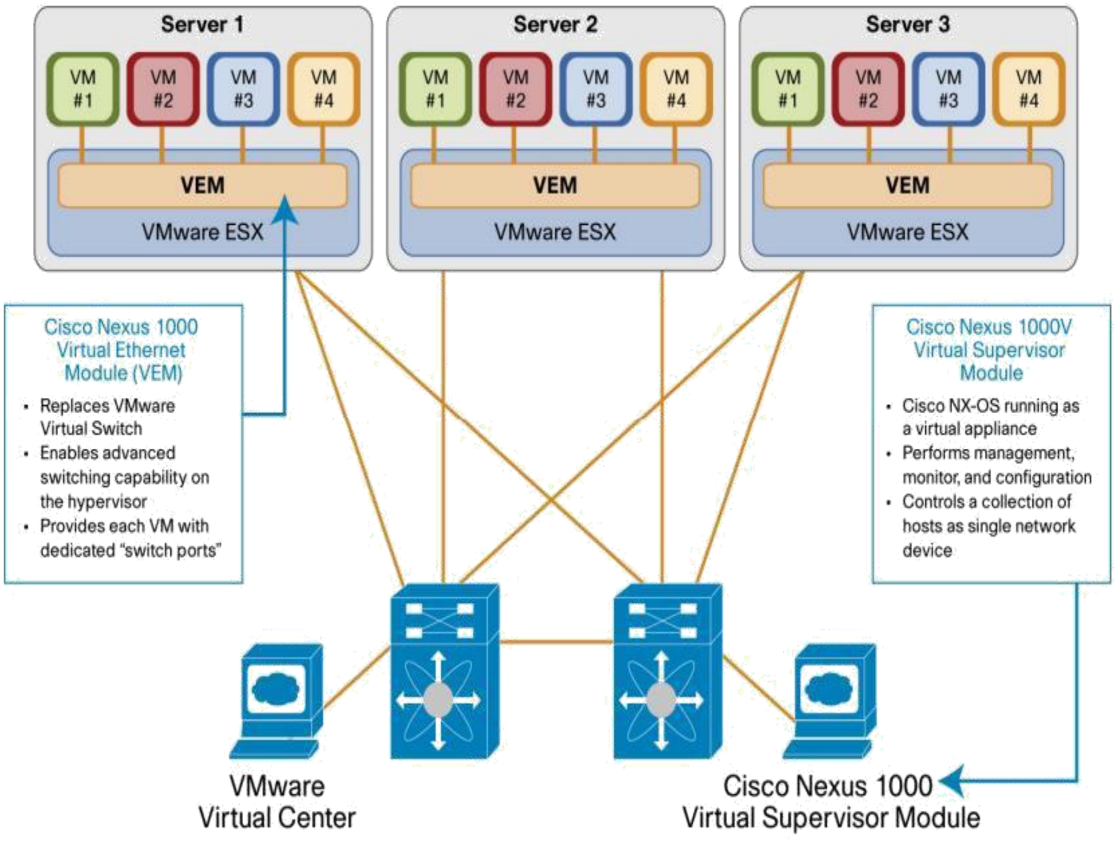

Server 1
Server 2
Server 3
VM #1
VM #2
VM #3
VM #4
VEM
VMware ESX
Cisco Nexus 1000 Virtual Ethernet Module (VEM)
Replaces VMware Virtual Switch
Enables advanced switching capability on the hypervisor
Provides each VM with dedicated "switch ports"
Cisco Nexus 1000V Virtual Supervisor Module
Cisco NX-OS running as a virtual appliance
Performs management, monitor, and configuration
Controls a collection of hosts as single network device
VMware Virtual Center
Cisco Nexus 1000 Virtual Supervisor Module

**Fig. 1.1 Cisco Nexus 1000v Architecture**

**I. VEM(Virtual Ethernet Module)-**

It is just a line card same as NIC, which runs inside the hypervisor. It relies on VSM used for Data forwarding. The VEM takes configuration information from the VSM and performs switching functions like-

- **Security:** Access Control Lists, Private VLAN, Cisco TrustSec architecture.

- **QOS (Quality of service)**

- **Monitoring:** In case of loss of communication with the Virtual Supervisor Module, the VEM has Nonstop Forwarding ability to keep on exchanging traffic dependent on last known setup. To put it plainly, the VEM provides advanced switching with data-center for the server virtualization environment. For example-NetFlow, RSPAN(Remote SPAN), SPAN(Switched Port Analyzer), it copies traffic over switches and forwards it out the SPAN. We can monitor the traffic via seeing incoming and outgoing traffic through a port.

**II. VSM(Virtual Supervisor Module)-**

VSM controls various VEMs as one coherent particular switch. Rather than physical line card modules, the Virtual Supervisor Module supports VEMs running in programming inside servers. It gives insight to switch and furthermore get to the single administration point. A couple of excess VSM can oversee upto 64 VEMs, which is equal to a chassis you exchanged with 64 line cards in it. VSM has 3 interfaces, Control, Management and Packet, and these are used in different operations when we configure our VSM and this is being layer 2 and layer 3. It comes in two mode, like one as VM that we can install on our VMware environment which we did in our project, and the second one is, it can be a VSA(Virtual Service Appliance) on a dedicated cisco Nexus 1010 or 1100 servers. All the configuration has been made on VSM and the VSM applies those configurations to the VEMs. We can see the tight integration between VSM and vCenter or VEM.

So instead of configuring every time inside the hypervisor on host, administrators can define configurations for instant use on all VEMs via VSM. As all the configurations and setups are performed via VSM and automatically transferred to all the VEMs

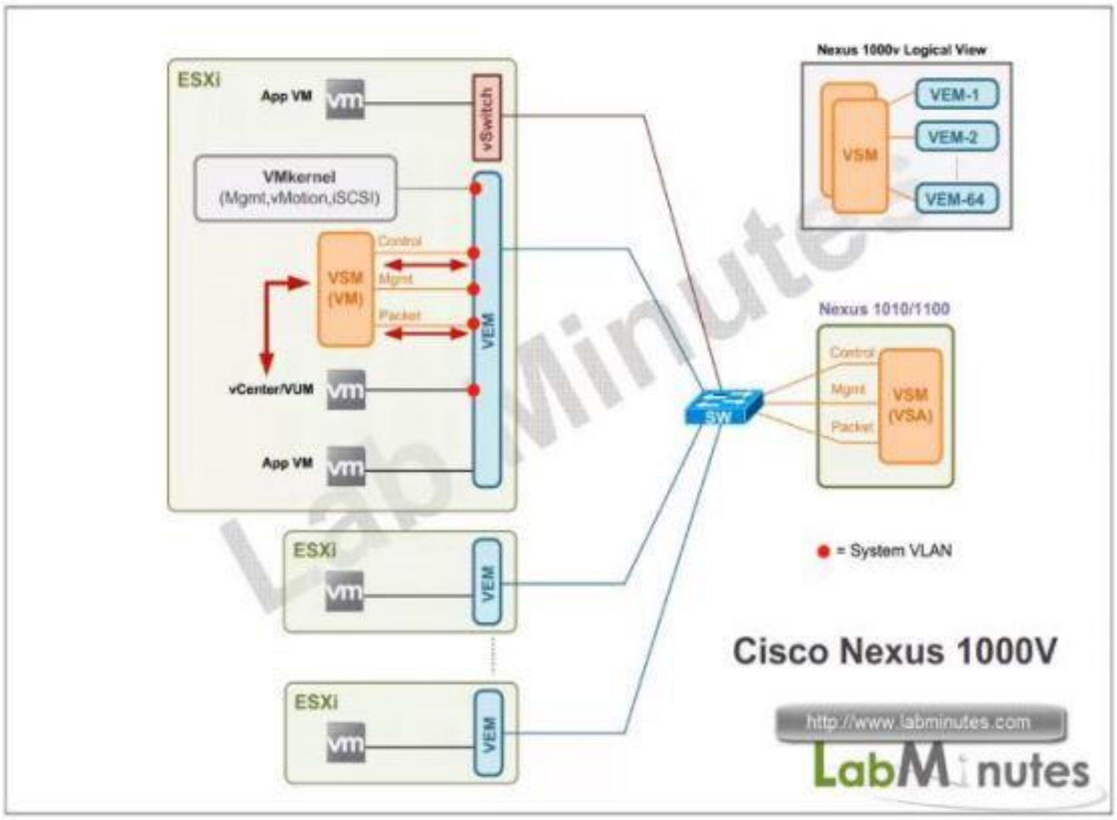


**Fig.1.2 Nexus 1000v VEM and VSM connection**

These are some benefits of Nexus 1000v Switch-

● **High availability**: Redundant and Synchronized Supervisors enable fast and stateful failover.

● **Scalability and Flexibility:** By configuring the ports by category we can scale up the number of ports. Including LAN, SAN or WAN a common application will be able to run all the areas of data center.

● **Manageability:** We can access or configure this switch through CLI (Command Line Interface), SNMOP(Simple Network Management Protocol) and XML or REST API. Much the same as the remainder of the Cisco Nexus family, the Cisco Nexus 1000V can likewise be overseen utilizing the far reaching devices of Cisco Data Center Network Manager.

## 1.2 Problem Statement

The main purpose of this project is to develop Python APIs for a Network Switch (Nexus 1000v) that can control the switch and eliminate the use of existing commands such as CURL. To access or configure any hardware switch, every network administrator has to deal with that in person but there are millions of switches and VLANs in our internet working from the server side, so instead of this Cisco implemented the virtual Switches which may run inside any computer where we access as many VLANs as we(customers) want.

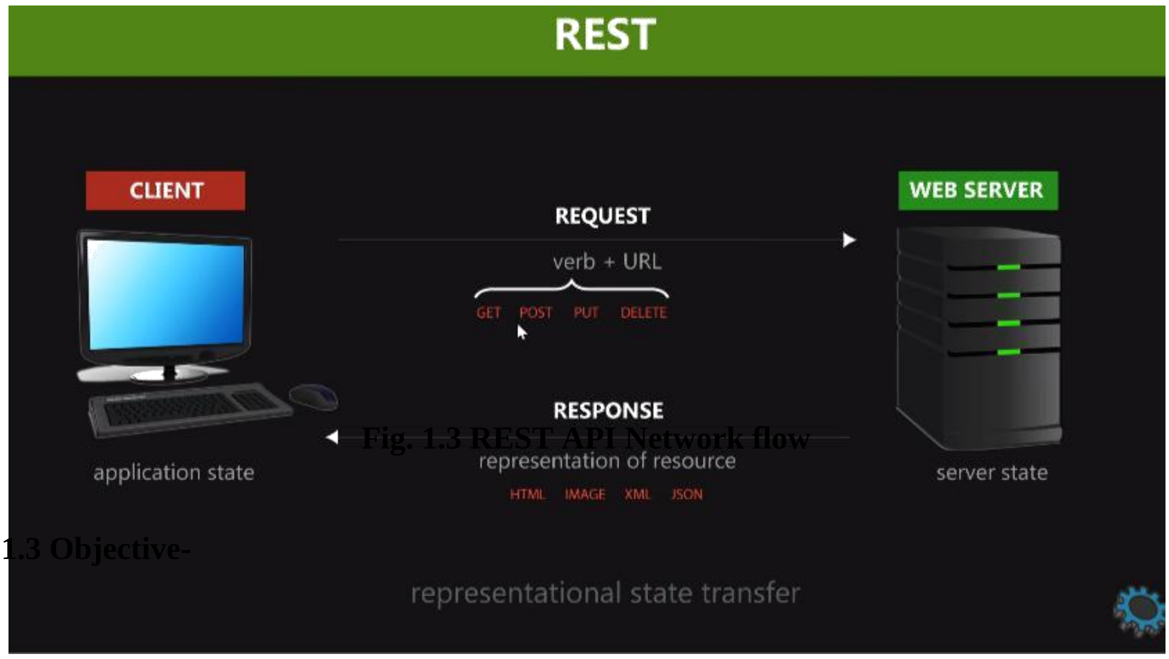


Fig. 1.3 REST API Network flow

## 1.3 Objective-

**Fig. 1.3 REST API Network flow**

As the scope in networking area goes high such as software defined networking, We can incorporate and control many system switches from a single dashboard and choice layers. In this project we were working on building REST APIs using Python language for accessing our network switch and managing Port Profiles, CLIs, VxLANs etc. Structuring an API that has a worked in verification mechanism that utilizes validation calls to make a safe session and keep unapproved clients from obtaining entrance. An API that clings to the standards of REST does not require the customer to know anything about the structure of API. Or maybe the server needs to give whatever data the customer needs to collaborate with the administration. The coding is done in Python language in Ubuntu environment which was installed inside the Oracle virtual box and then used.

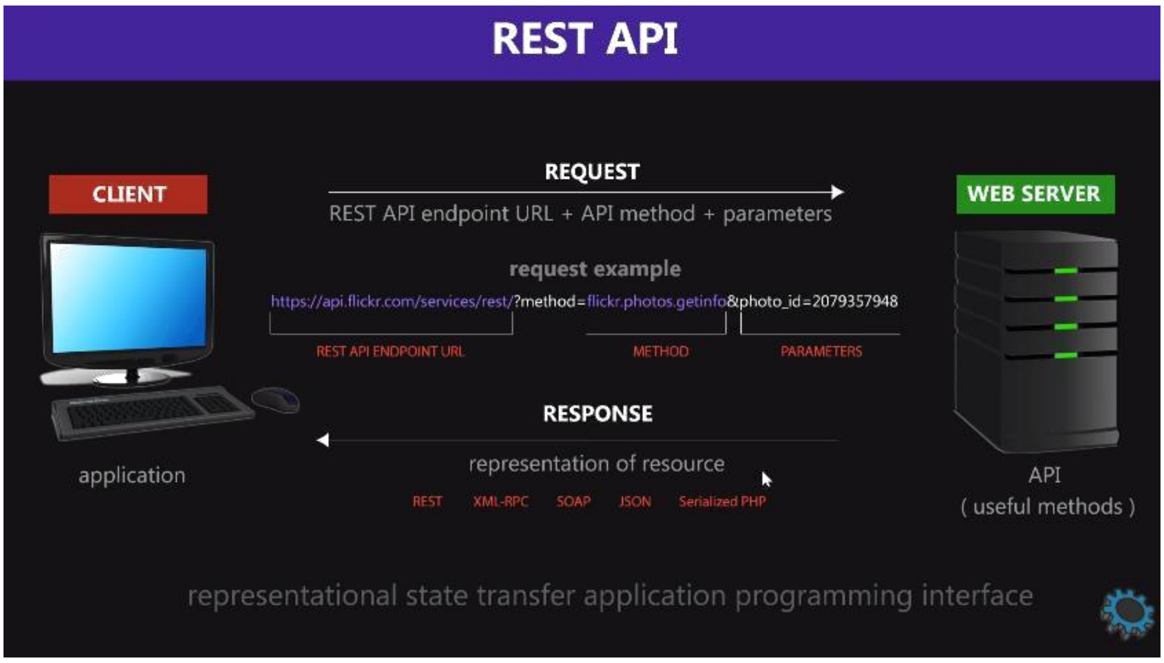

REST API
CLIENT
application
REQUEST
REST API endpoint URL + API method + parameters
request example
https://api.flickr.com/services/rest/?method=flickr.photos.getinfo&photo_id=2079357948
REST API ENDPOINT URL
METHOD
PARAMETERS
RESPONSE
representation of resource
REST
XML-RPC
SOAP
JSON
Serialized PHP
WEB SERVER
API
( useful methods )
representational state transfer application programming interface

**Fig. 1.4 REST API interface**

# Chapter 2. Literature Survey

## 2.1 Methodology-

Let’s take an example of HTML form, suppose you are a client, and want to access a webpage from abc.com website through your Web-Browser. But the browser does not know in the first place from where to access that web page you have been asking for.

So whenever we go to some URL it automatically transfers us to the server, from there all the information have been gathered. But all the date which is stored at the server side is in XML or HTML or JSON format, so it returns the data in an appropriate format regardless of whether client side is able to read that HTML or XML or JSON format or not. To resolve this problem, REST API came into existence, as it returns only the state objects of the data that you have asked for. So what makes it happen? The answer for this is- GET, POST PUT DELETE methods.

There is a well Organized method called CURD abbreviated as CREAT, UPDATE, READ and DELETE, which is done by POST, PUT, GET and DELETE respectively.

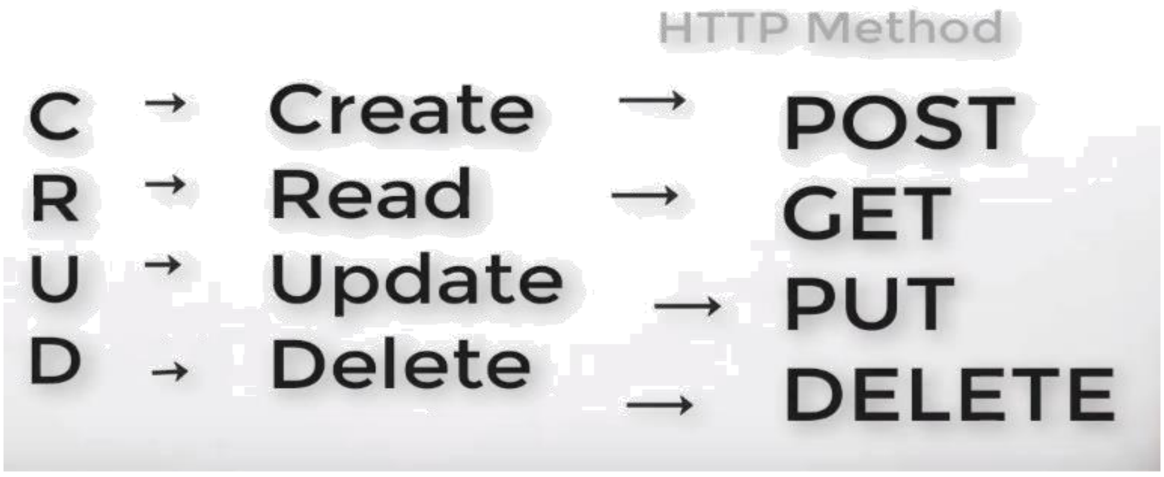


**Fig. 2.1 CURD**

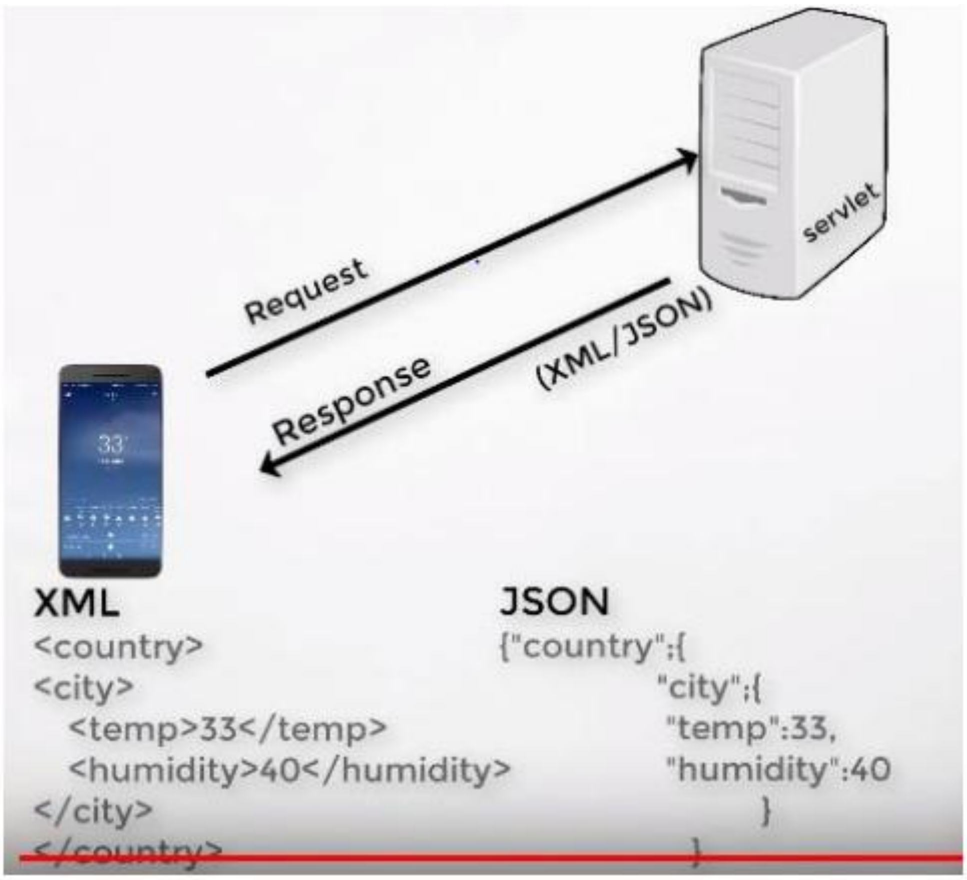
**GET-** It is READ-ONLY Method. We do a GET Request to do GET a message with a particular ID, it returns a representation in XML or JSON format with the HTTP response code of 200, which means result is [OK]. Whenever we want some date from the server we make a GET request to the server, in response, server understands that request and if you are an authorized user then it will send back the web page to the client. In GET request most the error happens from the server side not from the client side, Server side error code are 404[NOT FOUND] and 400[BAD Request].

So what does an API do is, it make cache for every GET Request we make, so that whenever we want to access the same webpage of date, it would not compute every time which create less latency. This caching only happens with the GET method because it doesn't change anything on the server side. So we can make as many GET requests as we want.

**POST-** POST is the method to create new resources on the webpage. It is not repeatable method, as we make the same POST request multiple times it may create duplicates of it. So to tackle this problem, what an API does is just make some safeguards for it, for example if you submitted a form on website, which is a POST request and then you hit refresh ,it will ask, "you've already submitted this data, want to continue?" it happens only to prevent the duplicates. Whenever a POST is done, normally the body is included in the requests that means sending along some sequence of bytes some data defining the object or record you are creating.

**PUT-** PUT is the method to update any information on the website. It is also repeatable method, as whenever you make the same call it updates the data every time.

**DELETE**- DELETE is also a save method as it removes the resource that a user want to delete from the website



```
XML
<country>
<city>
   <temp>33</temp>
   <humidity>40</humidity>
</city>
</country>
```

```
JSON
{"country":{
      "city":{
       "temp":33,
       "humidity":40
          }
      }
```

**Fig. 2.2 XML and JSON response from the server**.

## 2.2 Cisco Python Package-

Python is a simple to adapt, powerful programming language which has proficient high-level data structures and suitable approach to object oriented programming. Python's rich linguistic structure and dynamic composing, together with its deciphered nature, make it a perfect language for scripting and fast application improvement in numerous regions on generally stages. The Python interpreter and the broad standard library are openly accessible in source for every single significant stage from the Python website.

We have used Cisco Python SDK which provides a python package by which we can configure so many core network devices, and their modules such as Ports, Interfaces, Routes, ACL(Access Control List), VLANs(Virtual Area Network). Some source files and methods are for internal use and do not have embedded documentation

# Chapter 3. Command and Configurations

## 3.1 Installation of Nexus 1000v-

**Step 1.** Make sure that all of the VMware prerequisites have been met. ·

**Step 2.** Make sure that all of the Cisco Nexus 1000V prerequisites have been met. ·

**Step 3.** Read and follow the guidelines and limitations for the Cisco Nexus 1000V.

**Step 4.** Make topology decisions and gather any necessary information.

**Step 5.** Download the Cisco Nexus 1000V software.

**Step 6.** (Optional) Verify the authenticity of the Cisco Nexus 1000V image.

**Step 7.** Install the Virtual Supervisor Module (VSM) software from an ISO image, OVA image, or on a Cisco Nexus Cloud Services Platform. ·

**Step 8.** If you installed the VSM software on a CSP, proceed to the next step. If you installed the VSM software on a VM using an ISO or OVA image, you need to establish the SVS connection and configure the VM startup and shutdown parameters.

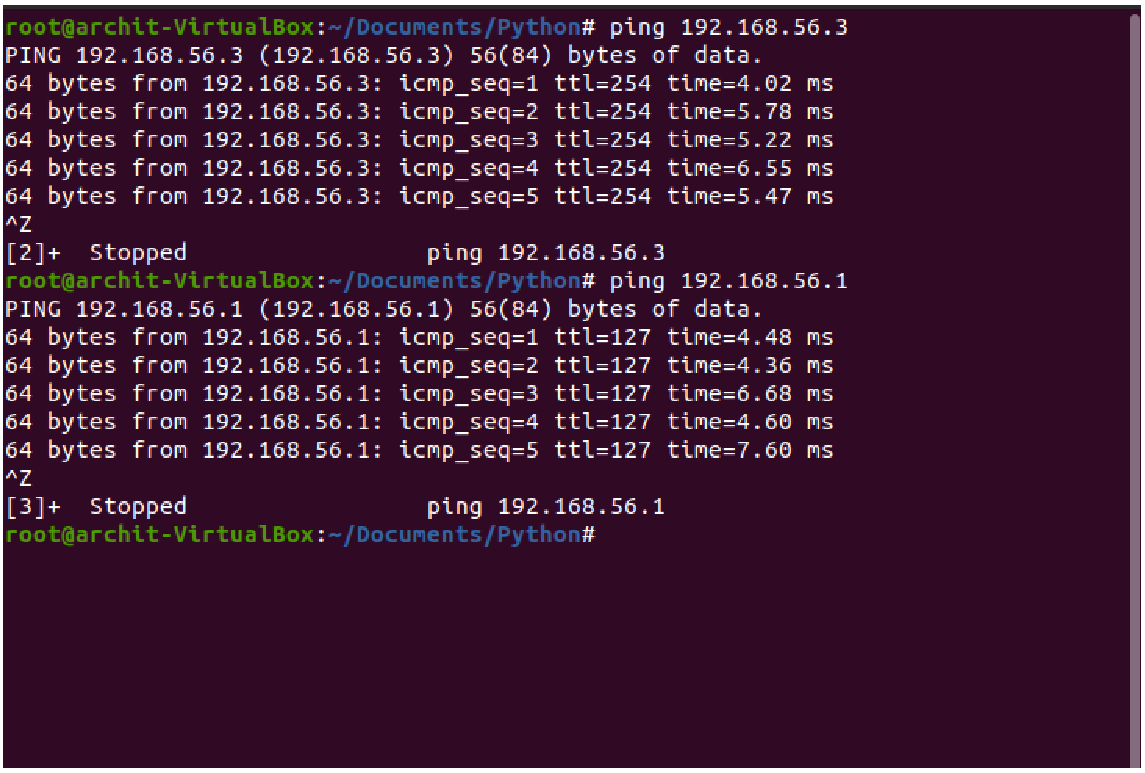
**Step 9.** Add the VEM hosts to the Distributed Virtual Switch.

**Step 10.** If you want to install the VEM software on a stateless ESXi host, proceed to the next step. Otherwise, install the VEM software using VUM, the Cisco Nexus 1000VCLI, or the VMware ESXi CLI.

**Step 11.** Install the VEM software on a stateless ESXi host

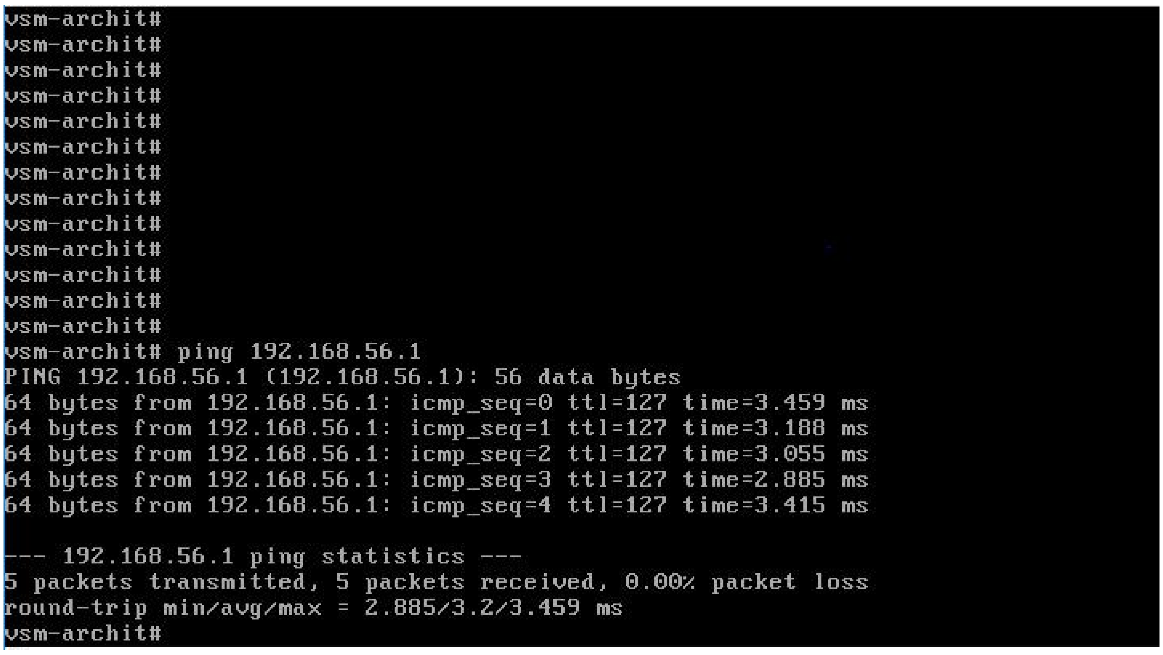
## 3.2 Connection among Windows, Ubuntu and Switch-

IP Addresses of these systems are- 192.168.56.1 (Windows), 192.168.56.2 (Ubuntu ), 192.168.56.3 (VSM)-



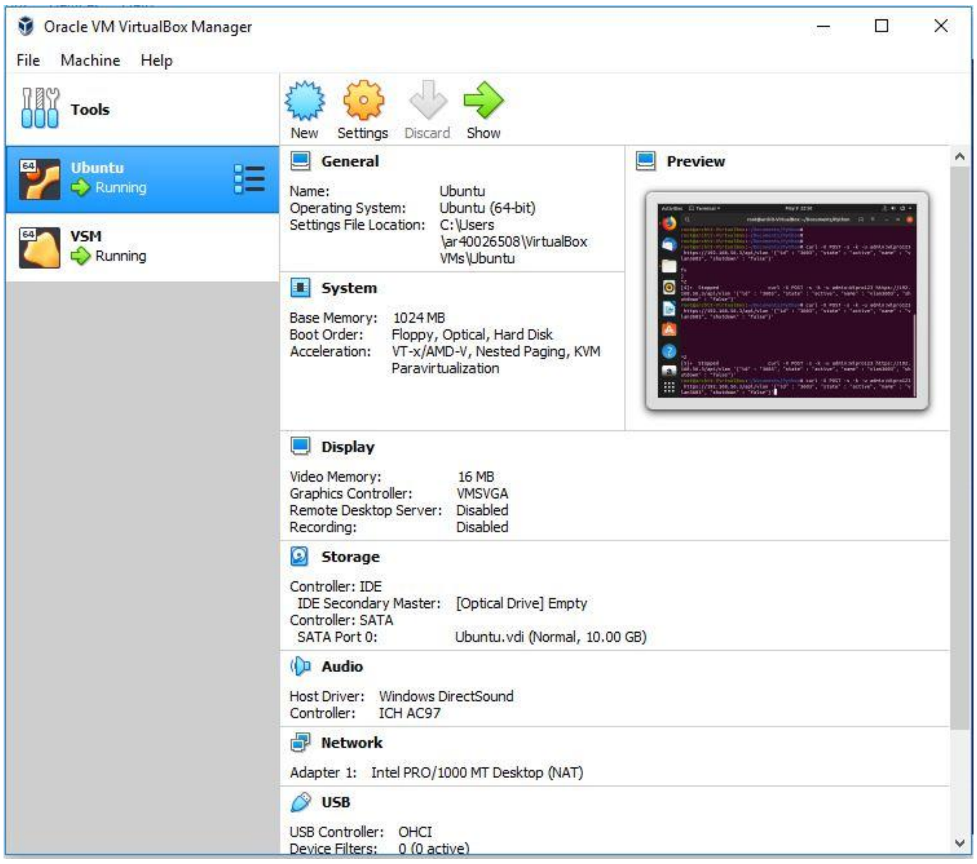
```
root@archit-VirtualBox:~/Documents/Python# ping 192.168.56.3
PING 192.168.56.3 (192.168.56.3) 56(84) bytes of data.
64 bytes from 192.168.56.3: icmp_seq=1 ttl=254 time=4.02 ms
64 bytes from 192.168.56.3: icmp_seq=2 ttl=254 time=5.78 ms
64 bytes from 192.168.56.3: icmp_seq=3 ttl=254 time=5.22 ms
64 bytes from 192.168.56.3: icmp_seq=4 ttl=254 time=6.55 ms
64 bytes from 192.168.56.3: icmp_seq=5 ttl=254 time=5.47 ms
^Z
[2]+  Stopped                 ping 192.168.56.3
root@archit-VirtualBox:~/Documents/Python# ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1) 56(84) bytes of data.
64 bytes from 192.168.56.1: icmp_seq=1 ttl=127 time=4.48 ms
64 bytes from 192.168.56.1: icmp_seq=2 ttl=127 time=4.36 ms
64 bytes from 192.168.56.1: icmp_seq=3 ttl=127 time=6.68 ms
64 bytes from 192.168.56.1: icmp_seq=4 ttl=127 time=4.60 ms
64 bytes from 192.168.56.1: icmp_seq=5 ttl=127 time=7.60 ms
^Z
[3]+  Stopped                 ping 192.168.56.1
root@archit-VirtualBox:~/Documents/Python#
```

**Fig. 3.1 Connection between Windows and Ubuntu**



```
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit#
vsm-archit# ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1): 56 data bytes
64 bytes from 192.168.56.1: icmp_seq=0 ttl=127 time=3.459 ms
64 bytes from 192.168.56.1: icmp_seq=1 ttl=127 time=3.188 ms
64 bytes from 192.168.56.1: icmp_seq=2 ttl=127 time=3.055 ms
64 bytes from 192.168.56.1: icmp_seq=3 ttl=127 time=2.885 ms
64 bytes from 192.168.56.1: icmp_seq=4 ttl=127 time=3.415 ms

--- 192.168.56.1 ping statistics ---
5 packets transmitted, 5 packets received, 0.00% packet loss
round-trip min/avg/max = 2.885/3.2/3.459 ms
vsm-archit#
```

**Fig. 3.2 Connection between Ubuntu and VSM**



General

Name: Ubuntu
Operating System: Ubuntu (64-bit)
Settings File Location: C:\Users \ar40026508\VirtualBox VMs\Ubuntu

System

Base Memory: 1024 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display

Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Secondary Master: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Ubuntu.vdi (Normal, 10.00 GB)

Audio

Host Driver: Windows DirectSound
Controller: ICH AC97

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB

USB Controller: OHCI
Device Filters: 0 (0 active)

**Fig. 3.3 System Configuration in VMware**

## 3.3 Port Profiles-

In Virtual switches Port Profiles are used to configure interfaces. It can be assigned to multiple interfaces to give them all the same configuration. If we make any changes to port profile, it directly propagated to the configuration of any interface assigned to it.

Same as any port works in a network device, Port profile also works for VMware vCenter server. In a port profile we can assign multiple vEthernet or Ethernet interfaces, because of the following reasons:

- Setting up a port configuration on the basis of policy.
- Large number of ports comes across a single policy.
- It supports both vEthernet and Ethernet.

## 3.4 User-

The API exposes methods that are user-related and that can be manipulated by users only.

## 3.5 VxLAN-

A VxLAN(Virtual Extensible LAN) creates LAN components by using MAC-in-UDP Encapsulation. Whenever an encapsulated MAC frames are sent over the network, each VEM is assigned an IP address which is used as the source IP address. A VxLAN ID is assigned within a port profile configuration of the Vnic, when the VM connects to the network. It also supports 3 different mode of operation as-Broadcast, MAC distribution transfer and Multicast.

## 3.6 Curl Commands-

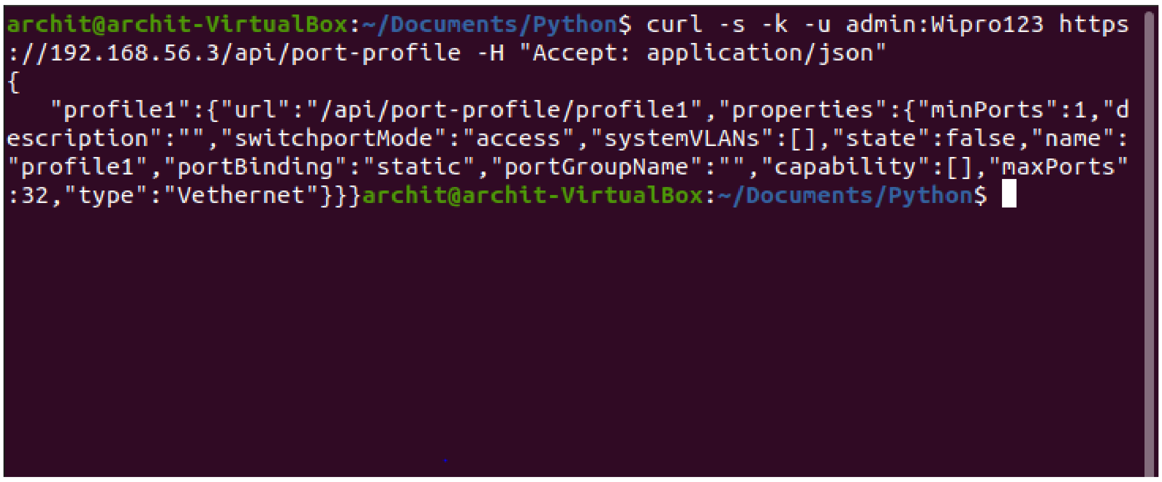
Curl commands allow us to GET / POST / PUT / DELETE data from any website from the command line itself. This is the best way test REST API. It gives the information of any kind of extension or protocols like HTTP,HTTPS, TELNET, FTP, SSH etc by transferring from or to a server. The REST API plug-in supports the JavaScript Object Notation (JSON) format for a response.

Some of the commands that we have used in our project are-

**>>curl [options] [http://]**

I. **JSON and XML response through CURL-**

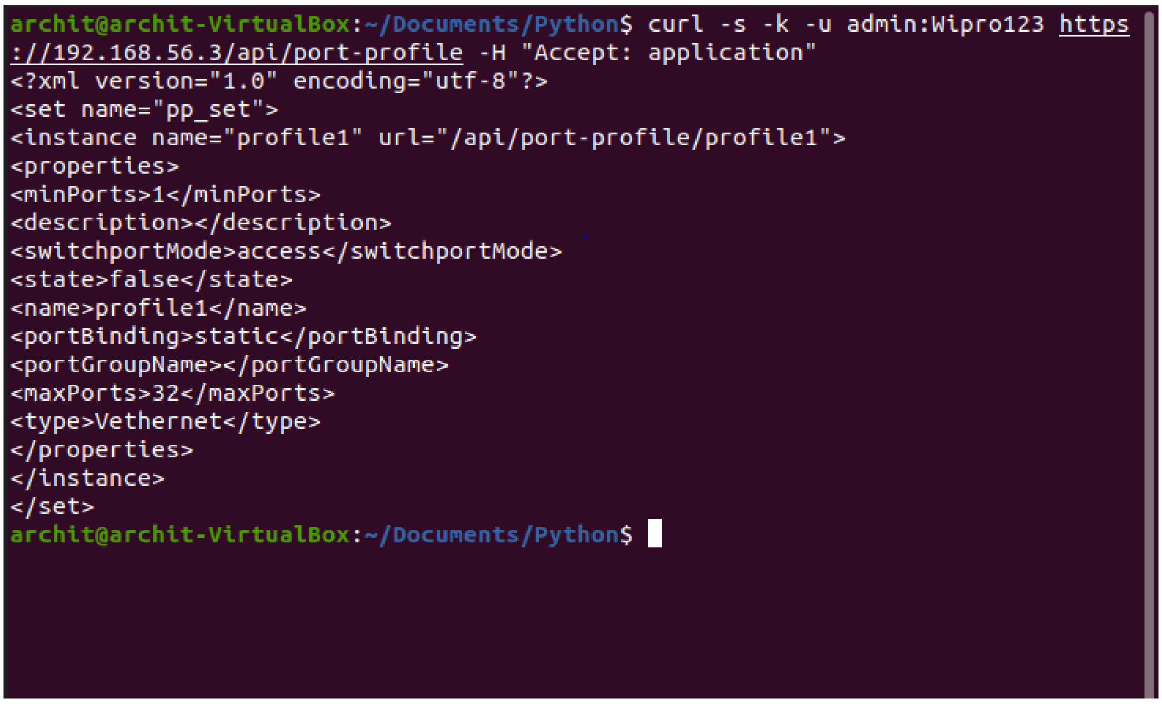
```
curl -u <user>:<password> <vsm-ip>/api/n1k/port-profile -H
"Accept:application/json”
```



```
archit@archit-VirtualBox:~/Documents/Python$ curl -s -k -u admin:Wipro123 https
://192.168.56.3/api/port-profile -H "Accept: application/json"
{
   "profile1":{"url":"/api/port-profile/profile1","properties":{"minPorts":1,"d
escription":"","switchportMode":"access","systemVLANs":[],"state":false,"name":
"profile1","portBinding":"static","portGroupName":"","capability":[],"maxPorts"
:32,"type":"Vethernet"}}}archit@archit-VirtualBox:~/Documents/Python$ 
```

**Fig 3.4 JSON Response through CURL**

The REST API plug-in also supports the XML format for a response. For XML response, specify **Accept: application/xml** in the HTTP header as:



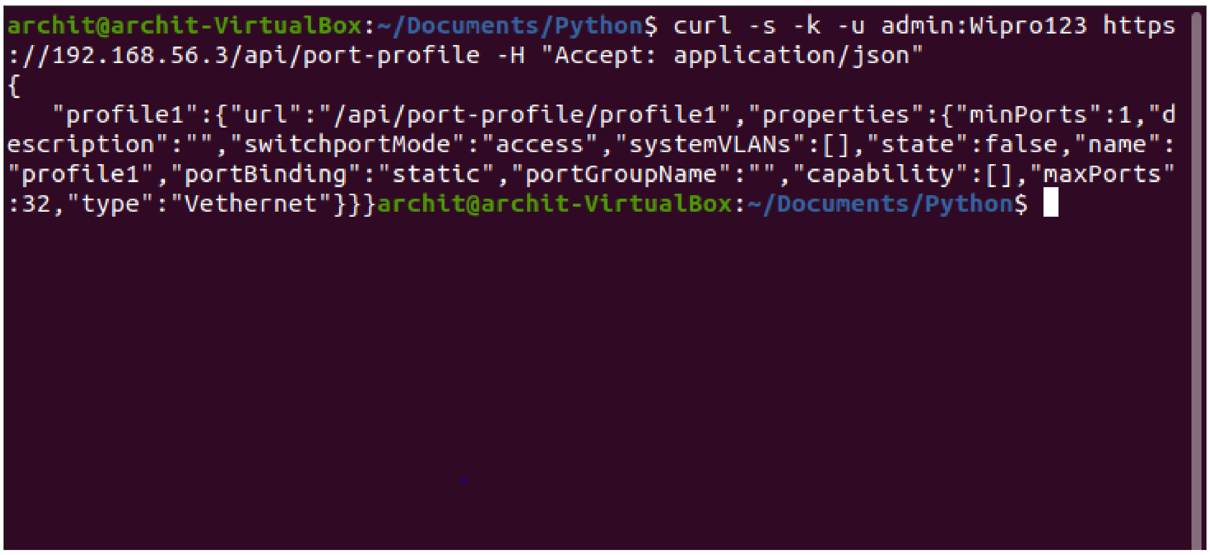
```
archit@archit-VirtualBox:~/Documents/Python$ curl -s -k -u admin:Wipro123 https
://192.168.56.3/api/port-profile -H "Accept: application"
<?xml version="1.0" encoding="utf-8"?>
<set name="pp_set">
<instance name="profile1" url="/api/port-profile/profile1">
<properties>
<minPorts>1</minPorts>
<description></description>
<switchportMode>access</switchportMode>
<state>false</state>
<name>profile1</name>
<portBinding>static</portBinding>
<portGroupName></portGroupName>
<maxPorts>32</maxPorts>
<type>Vethernet</type>
</properties>
</instance>
</set>
archit@archit-VirtualBox:~/Documents/Python$ 
```

**Fig 3.5 XML response using CURL**

## II. Port-Profile-

- **GET method to receive the Port-Profile information using CURL-**

  curl -u admin:password 192.168.56.3/api/n1k/port-profile/profile1 –H "Accept: application/json

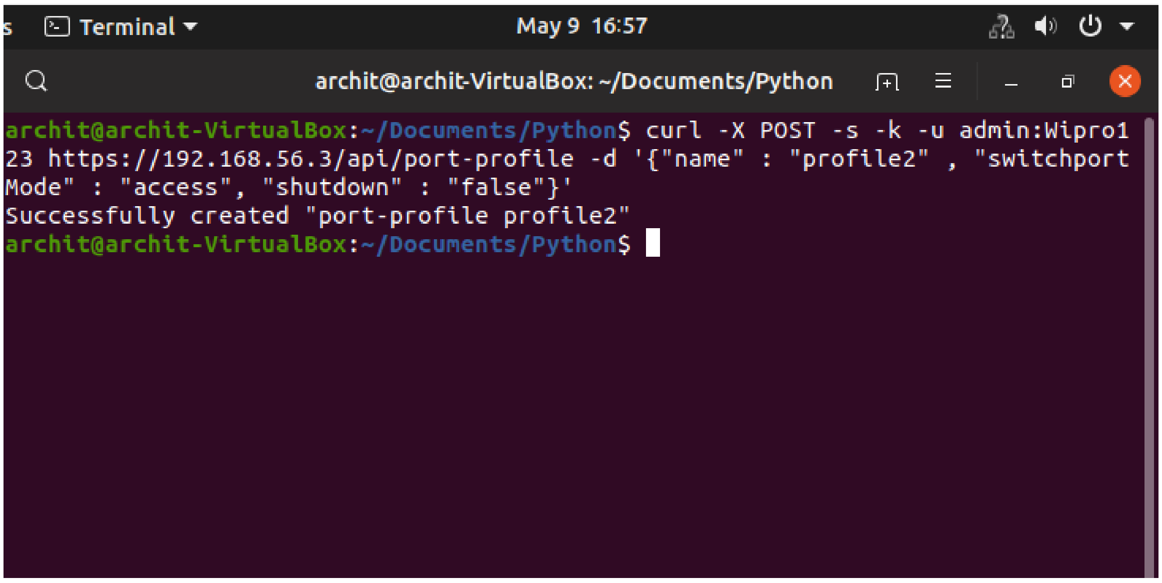


```
archit@archit-VirtualBox:~/Documents/Python$ curl -s -k -u admin:Wipro123 https
://192.168.56.3/api/port-profile -H "Accept: application/json"
{
   "profile1":{"url":"/api/port-profile/profile1","properties":{"minPorts":1,"d
escription":"","switchportMode":"access","systemVLANs":[],"state":false,"name":
"profile1","portBinding":"static","portGroupName":"","capability":[],"maxPorts"
:32,"type":"Vethernet"}}}archit@archit-VirtualBox:~/Documents/Python$ 
```

**Fig. 3.6 GET response using CURL**

- **POST method to create Port-Profile using CURL-**

  curl -X POST -u admin:password 192.168.56.3/api/n1k/port-profile -d '{"name" : "profile1", "switchportMode" : "access", "shutdown" : false}

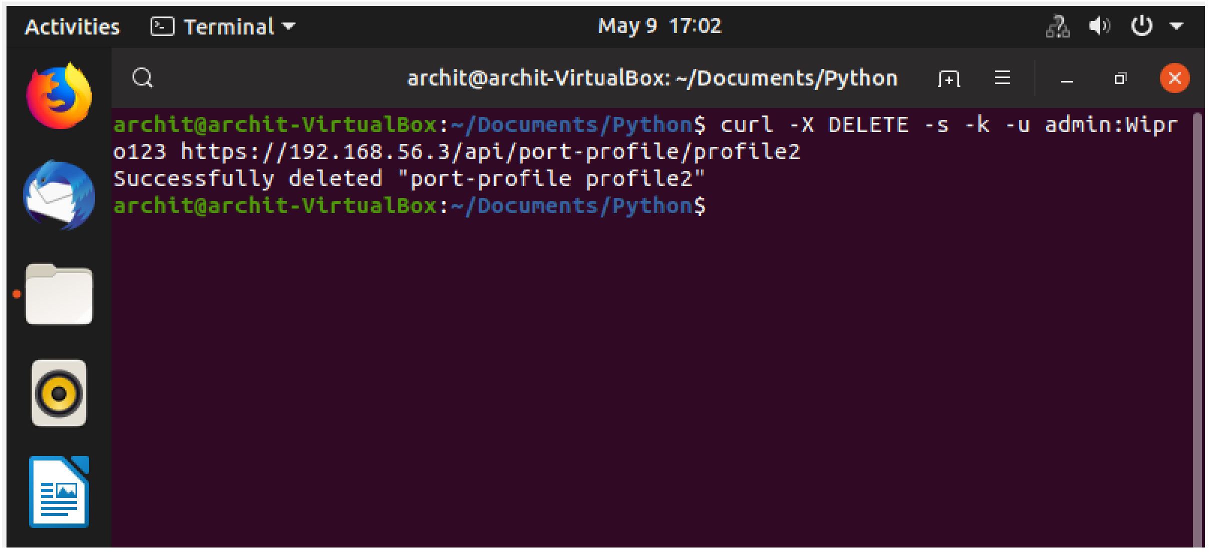


```
archit@archit-VirtualBox:~/Documents/Python$ curl -X POST -s -k -u admin:Wipro1
23 https://192.168.56.3/api/port-profile -d '{"name" : "profile2" , "switchport
Mode" : "access", "shutdown" : "false"}'
Successfully created "port-profile profile2"
archit@archit-VirtualBox:~/Documents/Python$ 
```

**Fig.3.7 POST response using CURL**

- **DELETE method to remove Port-Profile using CURL:**

  curl -u admin:password –X DELETE 192.168.56.3/api/port-profile/profile1

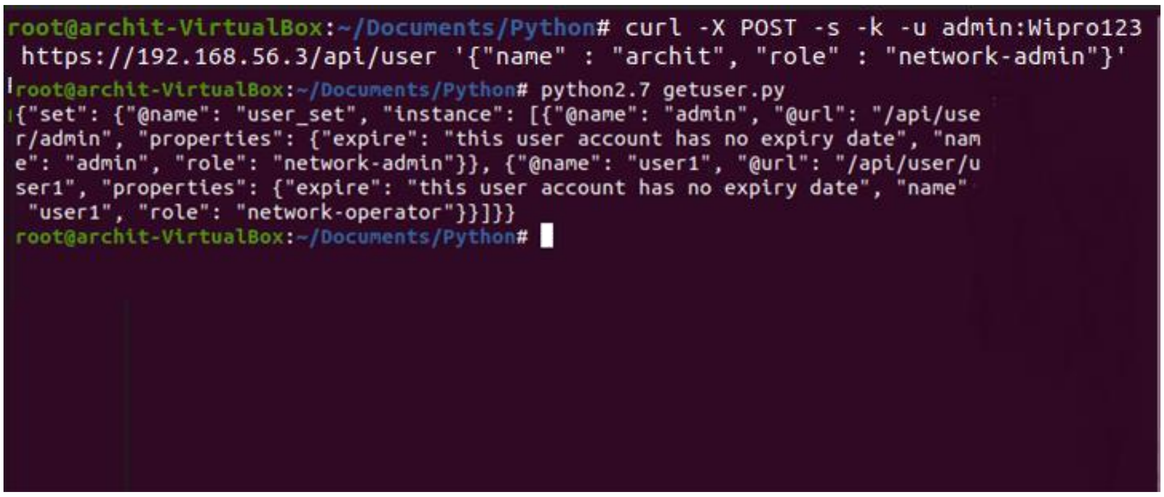


```
archit@archit-VirtualBox:~/Documents/Python$ curl -X DELETE -s -k -u admin:Wipr
o123 https://192.168.56.3/api/port-profile/profile2
Successfully deleted "port-profile profile2"
archit@archit-VirtualBox:~/Documents/Python$
```

**Fig. 3.8 DELETE Response using CURL**

**III. User-**

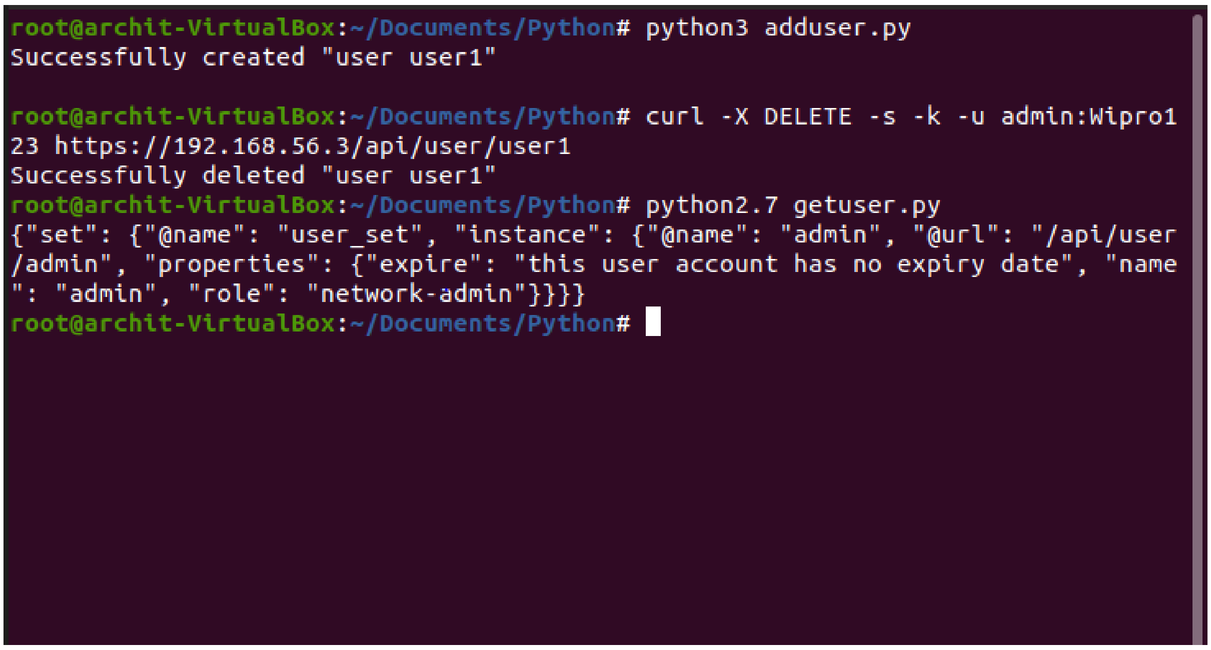
- **POST method to add new user information using CURL-**



```
root@archit-VirtualBox:~/Documents/Python# curl -X POST -s -k -u admin:Wipro123
 https://192.168.56.3/api/user '{"name" : "archit", "role" : "network-admin"}'
root@archit-VirtualBox:~/Documents/Python# python2.7 getuser.py
{"set": {"@name": "user_set", "instance": [{"@name": "admin", "@url": "/api/use
r/admin", "properties": {"expire": "this user account has no expiry date", "nam
e": "admin", "role": "network-admin"}}, {"@name": "user1", "@url": "/api/user/u
ser1", "properties": {"expire": "this user account has no expiry date", "name"
 "user1", "role": "network-operator"}}]}}
root@archit-VirtualBox:~/Documents/Python# 
```

**Fig. 3.9 Adding a new user using CURL**

- **DELETE method to remove user using CURL:**



```
root@archit-VirtualBox:~/Documents/Python# python3 adduser.py
Successfully created "user user1"

root@archit-VirtualBox:~/Documents/Python# curl -X DELETE -s -k -u admin:Wipro1
23 https://192.168.56.3/api/user/user1
Successfully deleted "user user1"
root@archit-VirtualBox:~/Documents/Python# python2.7 getuser.py
{"set": {"@name": "user_set", "instance": {"@name": "admin", "@url": "/api/user
/admin", "properties": {"expire": "this user account has no expiry date", "name
": "admin", "role": "network-admin"}}}}
root@archit-VirtualBox:~/Documents/Python# 
```

**Fig. 3.10 DELETE user using CURL**

# Chapter 4. Language Used

## 4.1 Python-

Python is a high level, general purpose and integrated language. It was designed by Guido van Rossum and first released in 1991. It follows OOPs approach and allows user to create logical code for large and small python based projects.

Python is a procedural object oriented language and is called dynamically-typed and garbage-collected. Due to its libraries it is often called ‘battery-included.

Python 2.0 released in 2000, introduced features like list comprehensions and a garbage collection system which is capable of collecting reference cycles. Python 3.0 which is released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.

**Libraries used:**

**1. Requests**

It is a standard library for making HTTP requests in Python. It is a very beautiful command, so that user can focus on interacting with services and consuming data in the application.

To use the package we first installed the package using the command

**>>Pip install requests**

Once, the library is installed we can use it use it.

**>>Import requests**

**2. JSON-**

JSON, is widely used for information exchange. It is used to pass the data around. It is used to deal with object literal syntax and it stands for Java Script Object Notation. It is easily readable.

To use the package first we have to install the package

**>>pip install json**

To use this command-

**>>import json**

**3. Xmltodict-**

It is a very convenient library which gives a feeling that one is working with XML while actually executing commands in JSON.

To install the package, command that were used:

**>>pip install xmltodict**

To use this command-

**>>import xmltodict**

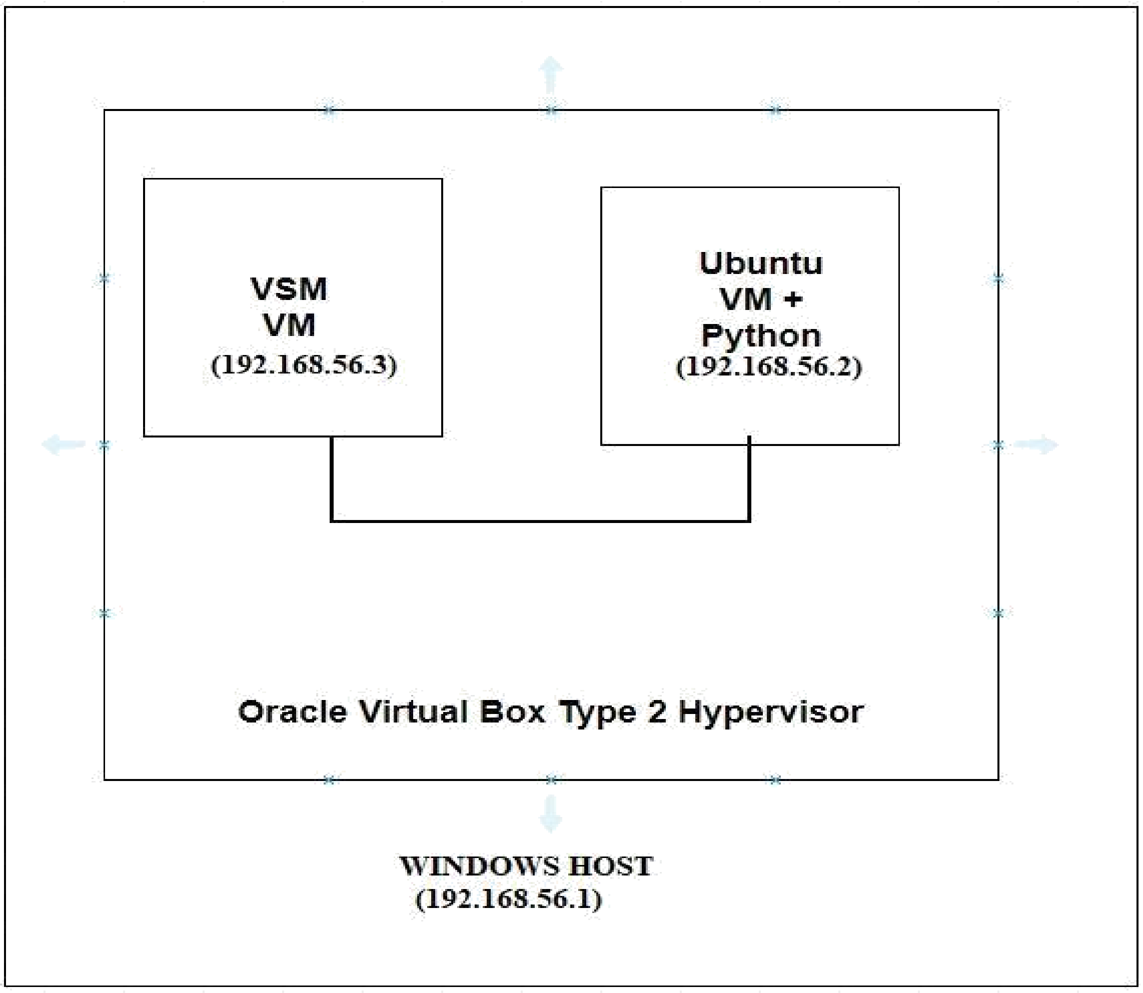


**Fig.4.1 Hardware layout of the system**

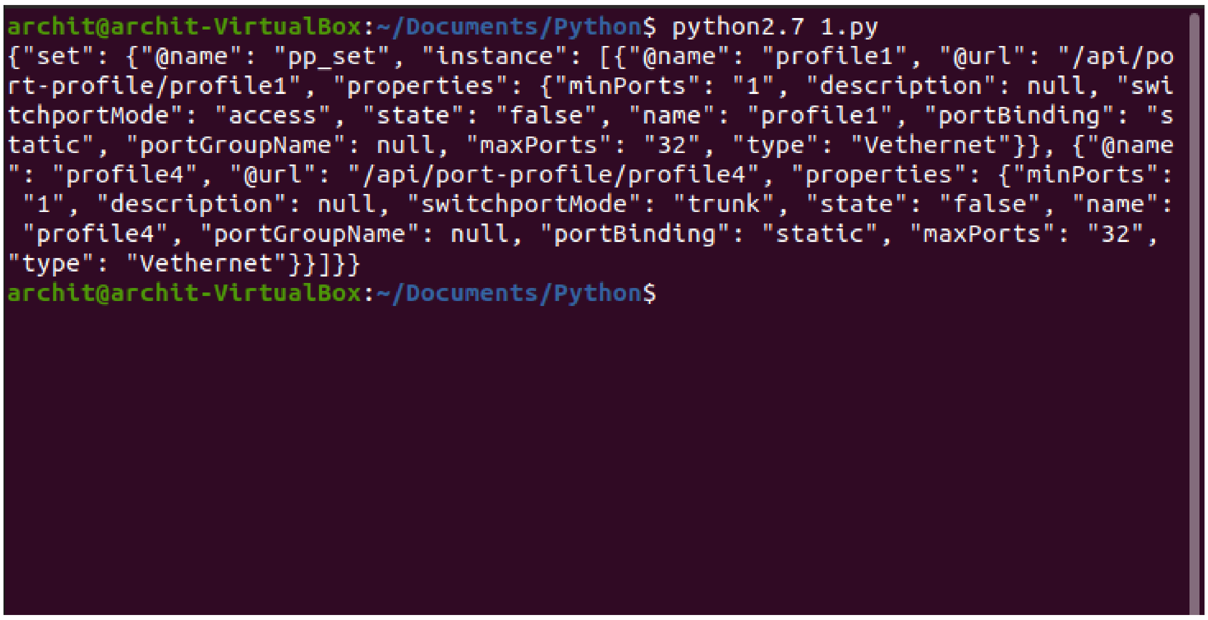
# Chapter 5. Results

## 5.1 Python API Response

**JSON Format-**

The REST API plug-in supports the JavaScript Object Notation (JSON) format for a response. To specify the JSON response format through Python API, use the following:

The following output shows the response received in the JSON format:



```
archit@archit-VirtualBox:~/Documents/Python$ python2.7 1.py
{"set": {"@name": "pp_set", "instance": [{"@name": "profile1", "@url": "/api/po
rt-profile/profile1", "properties": {"minPorts": "1", "description": null, "swi
tchportMode": "access", "state": "false", "name": "profile1", "portBinding": "s
tatic", "portGroupName": null, "maxPorts": "32", "type": "Vethernet"}}, {"@name
": "profile4", "@url": "/api/port-profile/profile4", "properties": {"minPorts":
 "1", "description": null, "switchportMode": "trunk", "state": "false", "name":
 "profile4", "portGroupName": null, "portBinding": "static", "maxPorts": "32",
"type": "Vethernet"}}]}}
archit@archit-VirtualBox:~/Documents/Python$
```

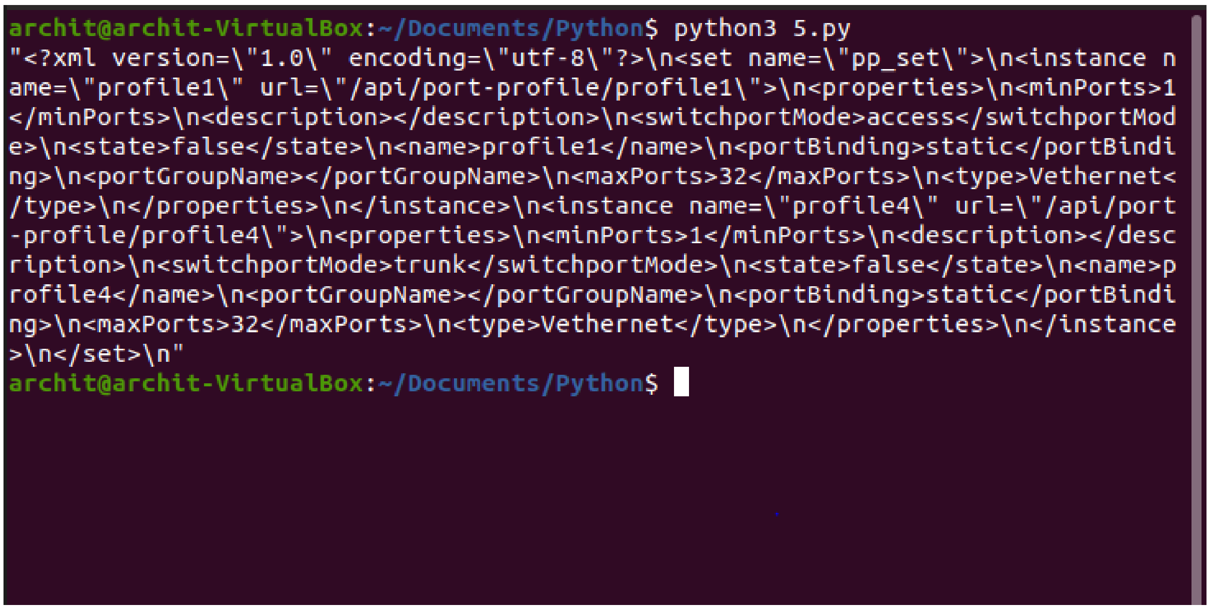
**Fig 5.1 JSON response using Python API**

**XML Format-**

The REST API plug-in supports the XML format for a response. For XML response, specify Accept: application/xml in the HTTP header

To specify the XML response format through Python

The following output shows the response received in the XML format



```
archit@archit-VirtualBox:~/Documents/Python$ python3 5.py
"<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<set name=\"pp_set\">\n<instance n
ame=\"profile1\" url=\"/api/port-profile/profile1\">\n<properties>\n<minPorts>1
</minPorts>\n<description></description>\n<switchportMode>access</switchportMod
e>\n<state>false</state>\n<name>profile1</name>\n<portBinding>static</portBindi
ng>\n<portGroupName></portGroupName>\n<maxPorts>32</maxPorts>\n<type>Vethernet<
/type>\n</properties>\n</instance>\n<instance name=\"profile4\" url=\"/api/port
-profile/profile4\">\n<properties>\n<minPorts>1</minPorts>\n<description></desc
ription>\n<switchportMode>trunk</switchportMode>\n<state>false</state>\n<name>p
rofile4</name>\n<portGroupName></portGroupName>\n<portBinding>static</portBindi
ng>\n<maxPorts>32</maxPorts>\n<type>Vethernet</type>\n</properties>\n</instance
>\n</set>\n"
archit@archit-VirtualBox:~/Documents/Python$
```

**Fig 5.2 XML Response using Python API**

## 5.2 Supported HTTP Methods-

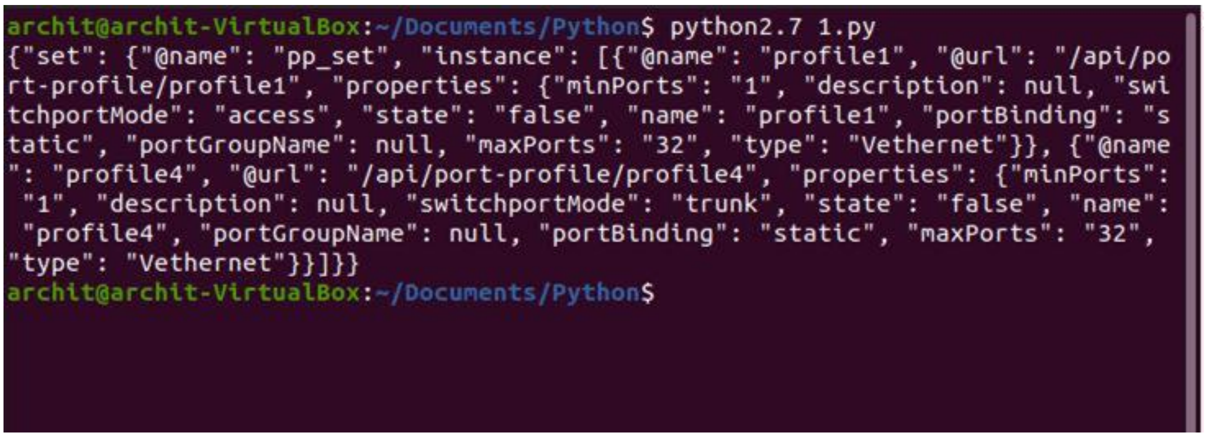
### I. Port-Profile-

#### GET Method-

The GET method lists the entities in a specific resource

The following is an output of the GET method in Python API:

.



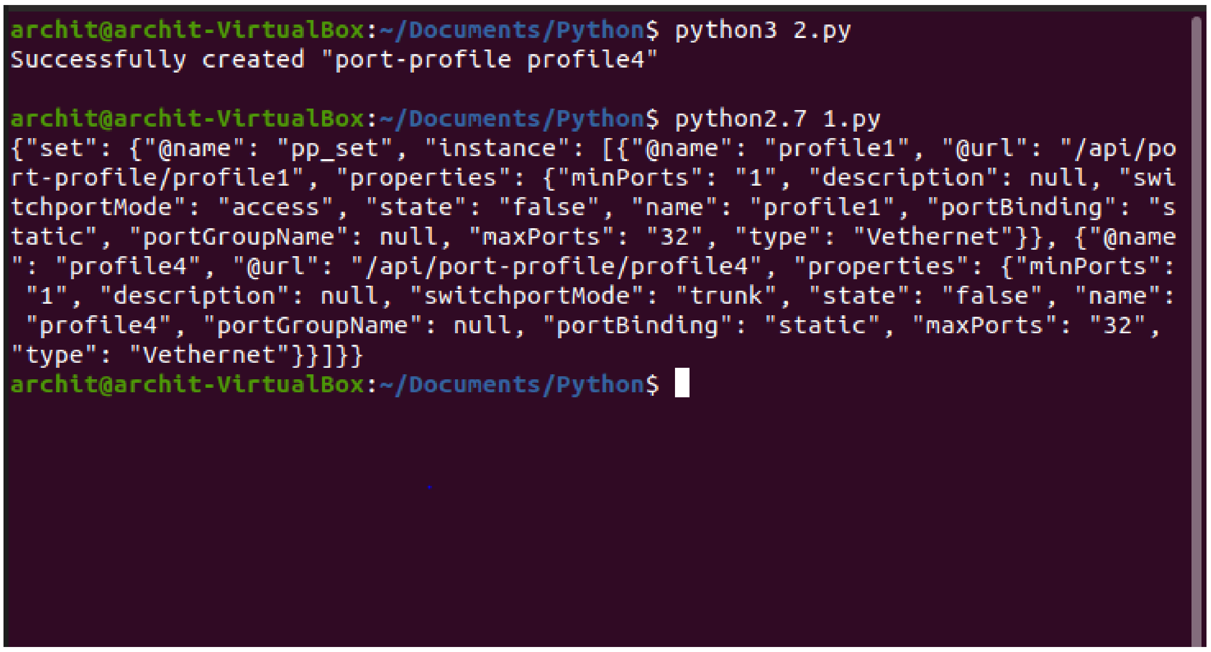
```
archit@archit-VirtualBox:~/Documents/Python$ python2.7 1.py
{"set": {"@name": "pp_set", "instance": [{"@name": "profile1", "@url": "/api/po
rt-profile/profile1", "properties": {"minPorts": "1", "description": null, "swi
tchportMode": "access", "state": "false", "name": "profile1", "portBinding": "s
tatic", "portGroupName": null, "maxPorts": "32", "type": "Vethernet"}}, {"@name
": "profile4", "@url": "/api/port-profile/profile4", "properties": {"minPorts":
 "1", "description": null, "switchportMode": "trunk", "state": "false", "name":
 "profile4", "portGroupName": null, "portBinding": "static", "maxPorts": "32",
"type": "Vethernet"}}]}}
archit@archit-VirtualBox:~/Documents/Python$
```

**Fig 5.3 GET Method Using Python API**

- **POST Method-**

The POST method creates a new instance of a resource or updates the identified instance.

The following is an output of the POST method in Python API:



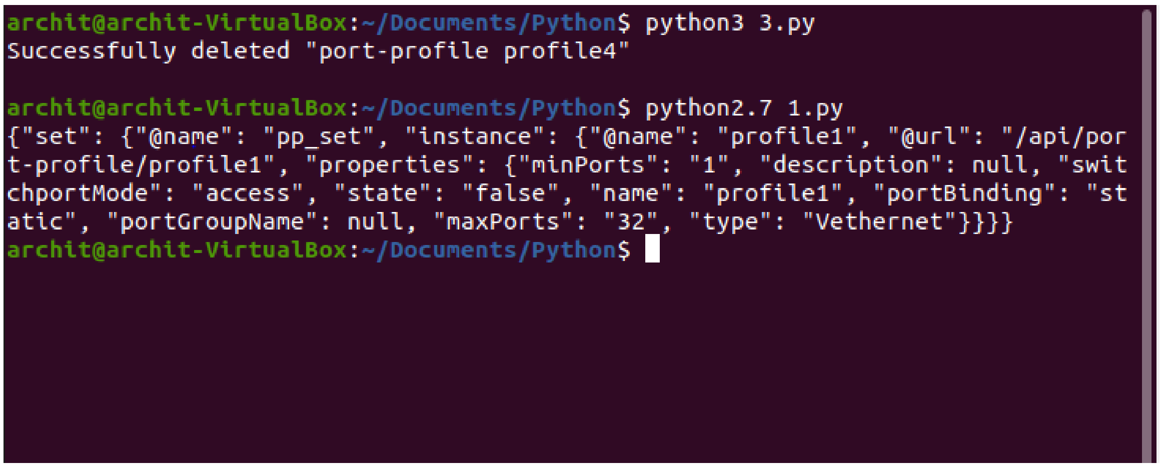
```
archit@archit-VirtualBox:~/Documents/Python$ python3 2.py
Successfully created "port-profile profile4"

archit@archit-VirtualBox:~/Documents/Python$ python2.7 1.py
{"set": {"@name": "pp_set", "instance": [{"@name": "profile1", "@url": "/api/po
rt-profile/profile1", "properties": {"minPorts": "1", "description": null, "swi
tchportMode": "access", "state": "false", "name": "profile1", "portBinding": "s
tatic", "portGroupName": null, "maxPorts": "32", "type": "Vethernet"}}, {"@name
": "profile4", "@url": "/api/port-profile/profile4", "properties": {"minPorts":
 "1", "description": null, "switchportMode": "trunk", "state": "false", "name":
 "profile4", "portGroupName": null, "portBinding": "static", "maxPorts": "32",
"type": "Vethernet"}}]}}
archit@archit-VirtualBox:~/Documents/Python$ 
```

**Fig 5.4 POST Port-Profile Method using Python API**

- **DELETE Method-**

The DELETE method deletes the specified instance.

The following is an output of the DELETE method in Python:



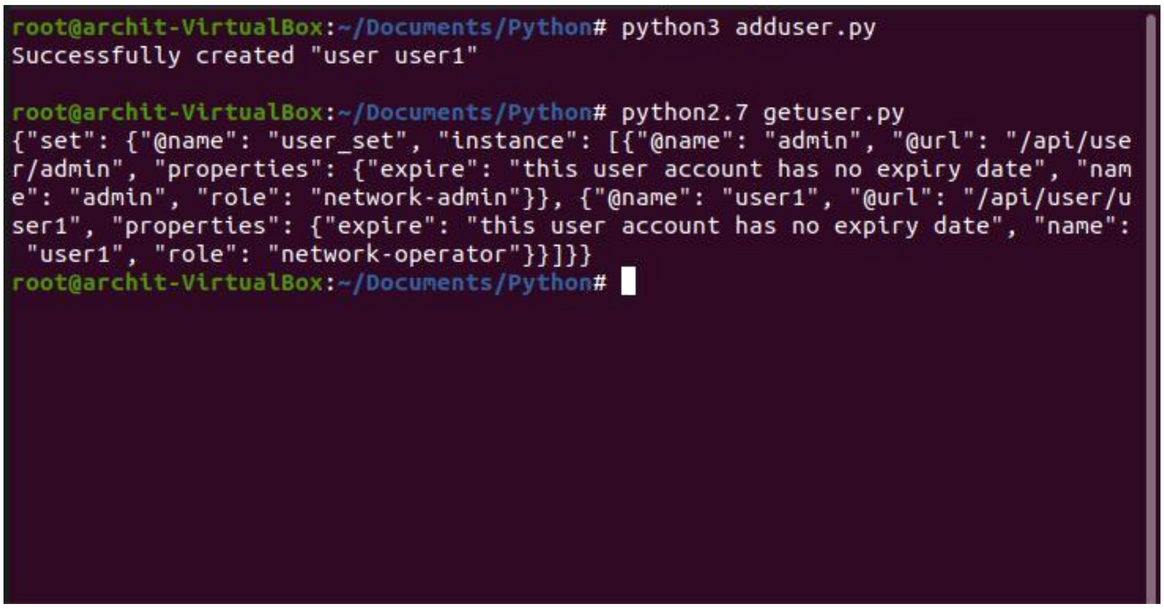
```
archit@archit-VirtualBox:~/Documents/Python$ python3 3.py
Successfully deleted "port-profile profile4"

archit@archit-VirtualBox:~/Documents/Python$ python2.7 1.py
{"set": {"@name": "pp_set", "instance": {"@name": "profile1", "@url": "/api/por
t-profile/profile1", "properties": {"minPorts": "1", "description": null, "swit
chportMode": "access", "state": "false", "name": "profile1", "portBinding": "st
atic", "portGroupName": null, "maxPorts": "32", "type": "Vethernet"}}}}
archit@archit-VirtualBox:~/Documents/Python$ 
```

**Fig 5.5 DELETE Port-Profile Method using Python API**

## II. User-

### POST Method-

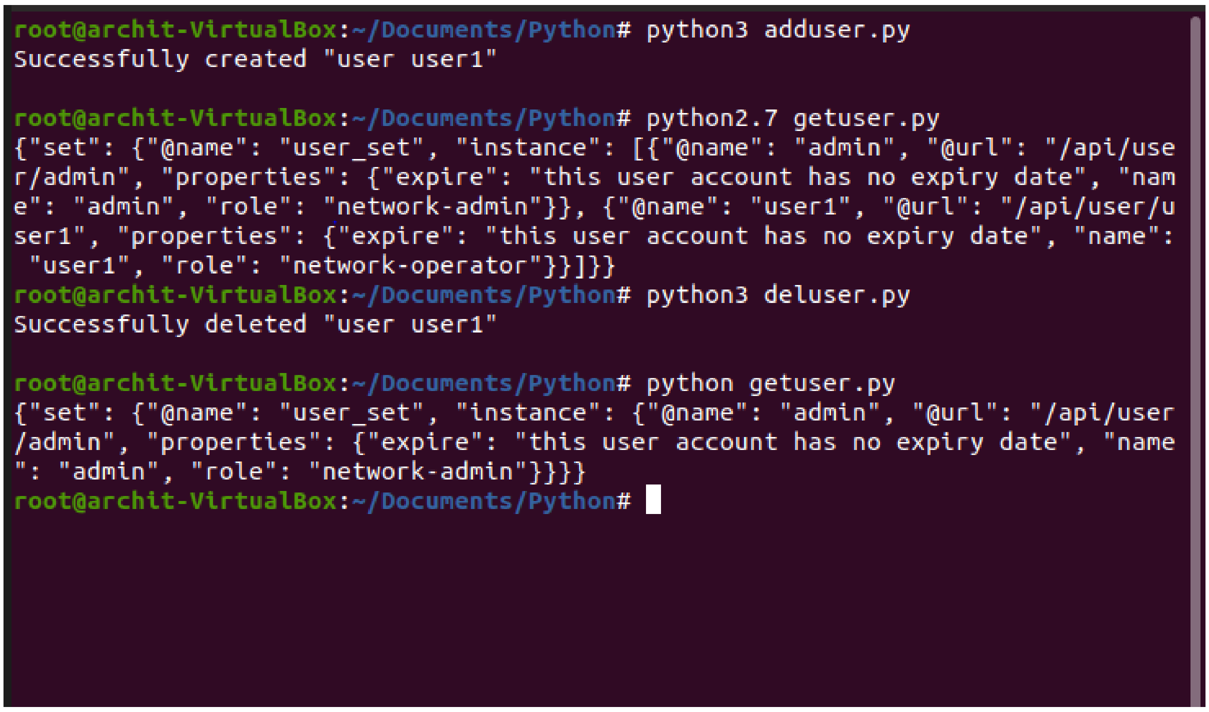
User can be added through python API. The expiration and the role can also be specified for the user in the network switch. The following is the output for adding the user-



```
root@archit-VirtualBox:~/Documents/Python# python3 adduser.py
Successfully created "user user1"

root@archit-VirtualBox:~/Documents/Python# python2.7 getuser.py
{"set": {"@name": "user_set", "instance": [{"@name": "admin", "@url": "/api/use
r/admin", "properties": {"expire": "this user account has no expiry date", "nam
e": "admin", "role": "network-admin"}}, {"@name": "user1", "@url": "/api/user/u
ser1", "properties": {"expire": "this user account has no expiry date", "name":
 "user1", "role": "network-operator"}}]}}
root@archit-VirtualBox:~/Documents/Python# 
```

**Fig. 5.6 Adding User using Python API**

**DELETE method:**

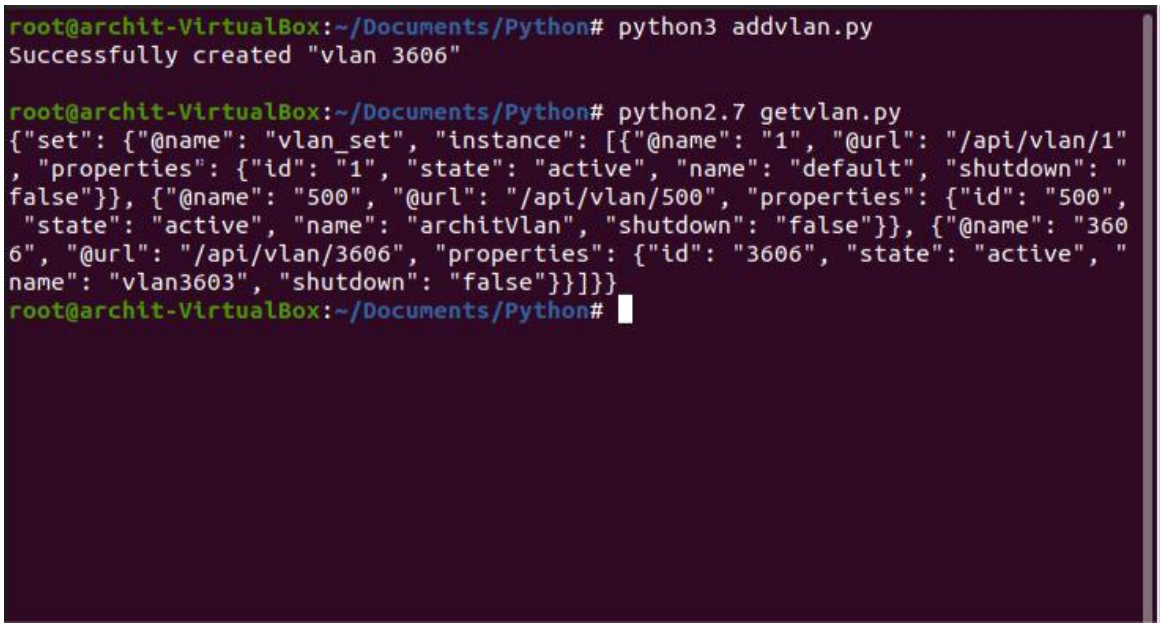


```
root@archit-VirtualBox:~/Documents/Python# python3 adduser.py
Successfully created "user user1"

root@archit-VirtualBox:~/Documents/Python# python2.7 getuser.py
{"set": {"@name": "user_set", "instance": [{"@name": "admin", "@url": "/api/use
r/admin", "properties": {"expire": "this user account has no expiry date", "nam
e": "admin", "role": "network-admin"}}, {"@name": "user1", "@url": "/api/user/u
ser1", "properties": {"expire": "this user account has no expiry date", "name":
 "user1", "role": "network-operator"}}]}}
root@archit-VirtualBox:~/Documents/Python# python3 deluser.py
Successfully deleted "user user1"

root@archit-VirtualBox:~/Documents/Python# python getuser.py
{"set": {"@name": "user_set", "instance": {"@name": "admin", "@url": "/api/user
/admin", "properties": {"expire": "this user account has no expiry date", "name
": "admin", "role": "network-admin"}}}}
root@archit-VirtualBox:~/Documents/Python# 
```

**Fig. 5.7 Deleting User using Python API**

**III. VLAN-**

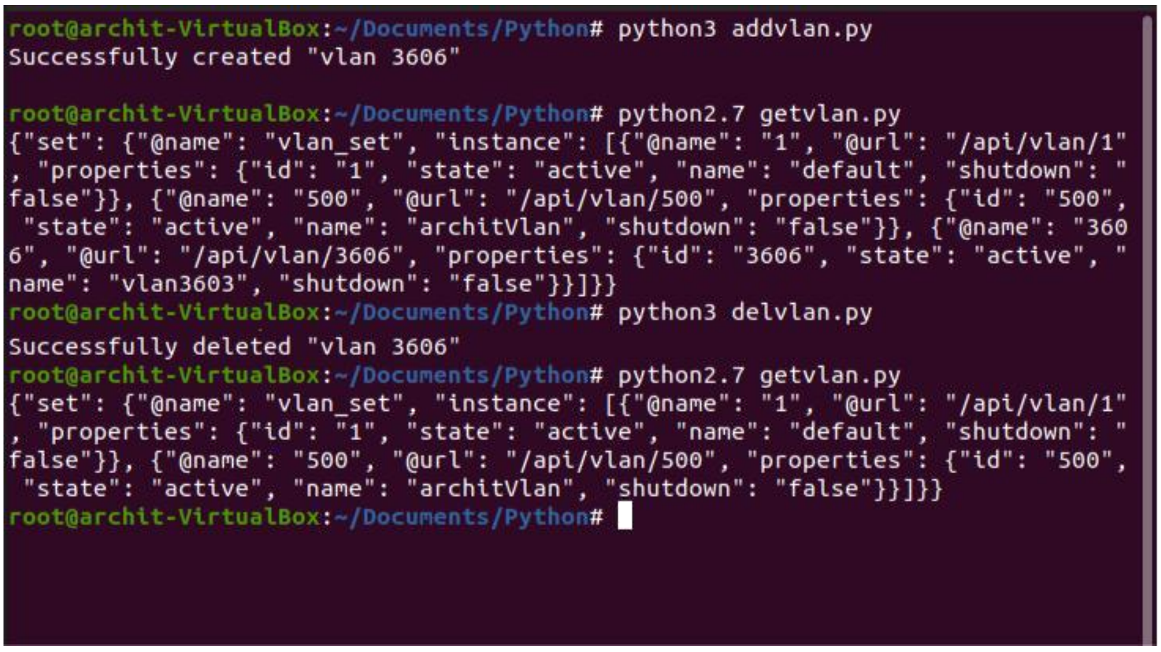
**POST Method-** It creates new VLANs to the users.



```
root@archit-VirtualBox:~/Documents/Python# python3 addvlan.py
Successfully created "vlan 3606"

root@archit-VirtualBox:~/Documents/Python# python2.7 getvlan.py
{"set": {"@name": "vlan_set", "instance": [{"@name": "1", "@url": "/api/vlan/1"
, "properties": {"id": "1", "state": "active", "name": "default", "shutdown": "
false"}}, {"@name": "500", "@url": "/api/vlan/500", "properties": {"id": "500",
 "state": "active", "name": "architVlan", "shutdown": "false"}}, {"@name": "360
6", "@url": "/api/vlan/3606", "properties": {"id": "3606", "state": "active", "
name": "vlan3603", "shutdown": "false"}}]}}
root@archit-VirtualBox:~/Documents/Python# 
```

**Fig. 5.8 Creating a new VLAN using Python API**

**DELETE Method-** It removes specific VLANs that a user wants.



```
root@archit-VirtualBox:~/Documents/Python# python3 addvlan.py
Successfully created "vlan 3606"

root@archit-VirtualBox:~/Documents/Python# python2.7 getvlan.py
{"set": {"@name": "vlan_set", "instance": [{"@name": "1", "@url": "/api/vlan/1"
, "properties": {"id": "1", "state": "active", "name": "default", "shutdown": "
false"}}, {"@name": "500", "@url": "/api/vlan/500", "properties": {"id": "500",
 "state": "active", "name": "architVlan", "shutdown": "false"}}, {"@name": "360
6", "@url": "/api/vlan/3606", "properties": {"id": "3606", "state": "active", "
name": "vlan3603", "shutdown": "false"}}]}}
root@archit-VirtualBox:~/Documents/Python# python3 delvlan.py
Successfully deleted "vlan 3606"
root@archit-VirtualBox:~/Documents/Python# python2.7 getvlan.py
{"set": {"@name": "vlan_set", "instance": [{"@name": "1", "@url": "/api/vlan/1"
, "properties": {"id": "1", "state": "active", "name": "default", "shutdown": "
false"}}, {"@name": "500", "@url": "/api/vlan/500", "properties": {"id": "500",
 "state": "active", "name": "architVlan", "shutdown": "false"}}]}}
root@archit-VirtualBox:~/Documents/Python# 
```

**Fig. 5.9 Deleting VLAN using Python API**

# Chapter 6. Conclusion and Future Scope

## 6.1 Conclusion-

This Python API has the job of connecting the operating system to the network switches .The project work proposes a new aspect and approach to fasten the procedure of accessing the networks switches.

## 6.2 Future Scope-

In future many Python scripts can be developed for Cisco Nexus 1000v with capabilities of:

- Run a script to verify configuration on switch
- boot-up. Back up a configuration.
- Proactive congestion management by monitoring and responding to buffer utilization characteristics.
- Ability to perform a job at a specific time interval.
- Programmatic access to the switch command line interface (CLI) to perform various tasks.

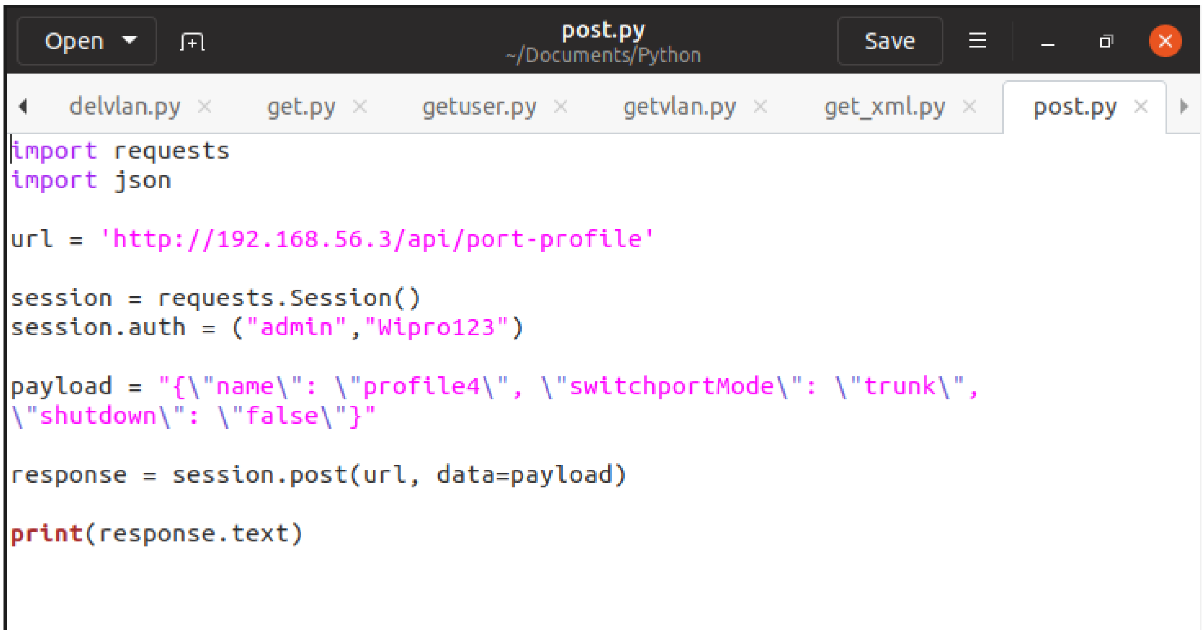
# Chapter 7. References

- https://www.cisco.com/c/dam/global/en_sg/trainingevents/datacentertechbyte/assets/pdfs/n1000_datasheet.pdf
- https://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/fundamentals-of-vxlan.html
- https://www.cisco.com/c/en/us/support/switches/nexus-1000v-switch-vmware-vsphere/products-installation-and-configuration-guides-list.html
- https://www.cisco.com/c/en/us/support/switches/nexus-1000v-switch-vmware-vsphere/products-installation-guides-list.html
- https://www.youtube.com/watch?v=pgksWMUBmcI

# Chapter 8. Appendices

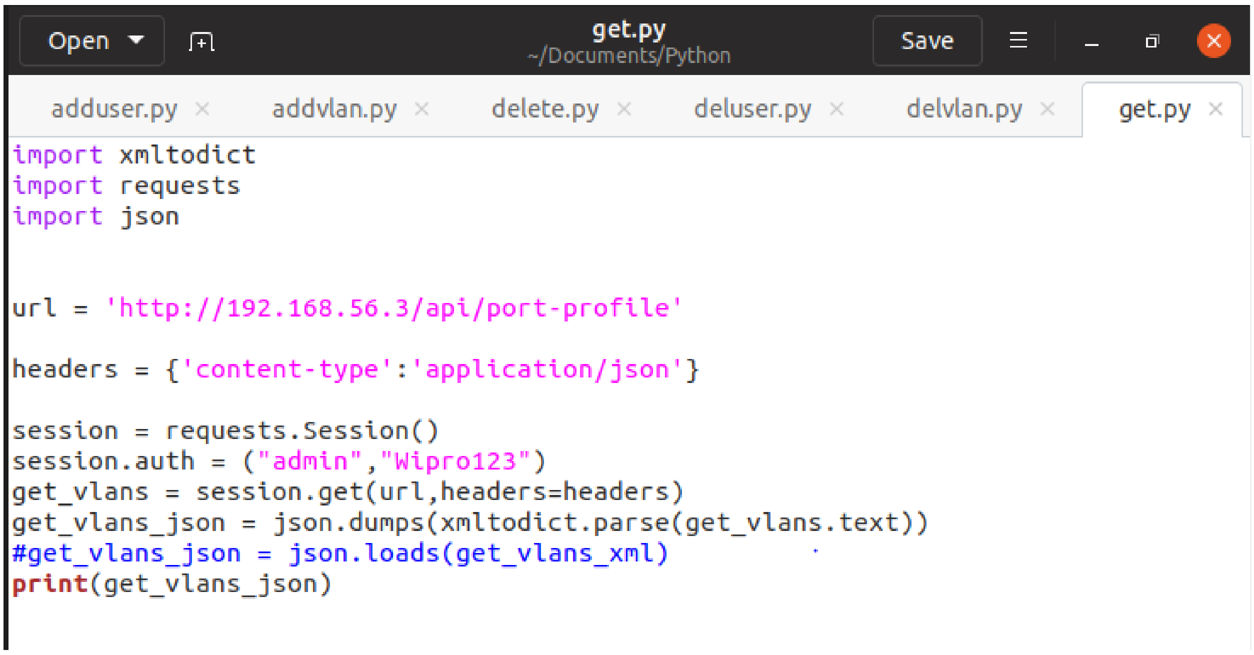
## 8.1 Port-Profile

1. create port-profile



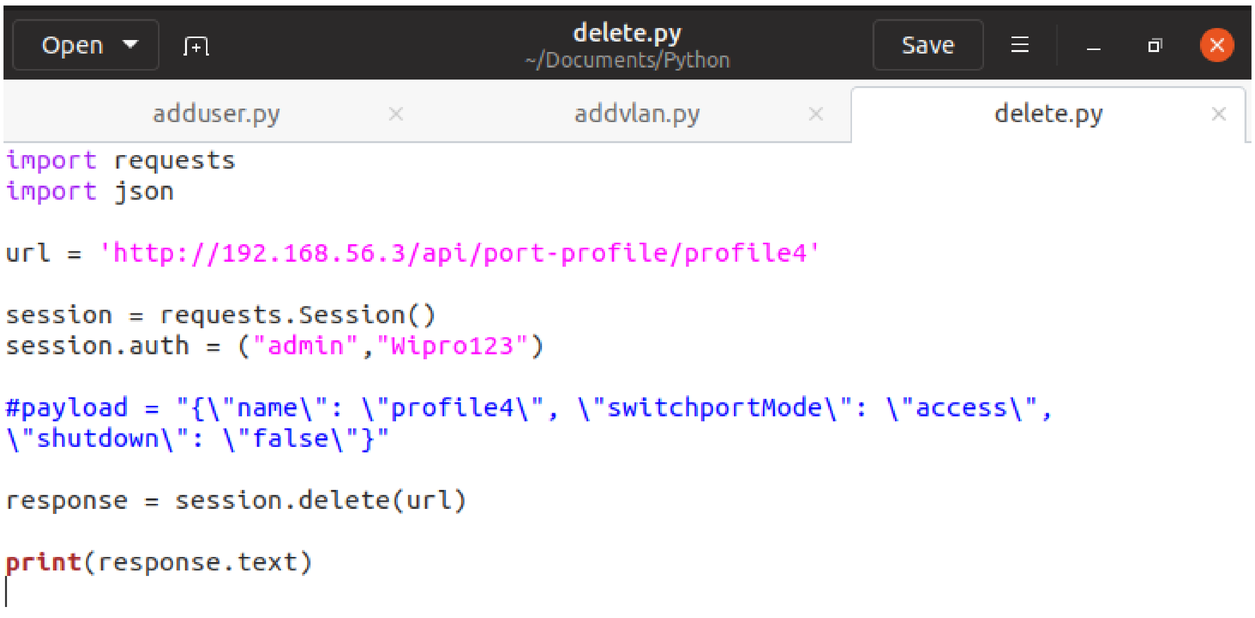
```
import requests
import json

url = 'http://192.168.56.3/api/port-profile'

session = requests.Session()
session.auth = ("admin","Wipro123")

payload = "{\"name\": \"profile4\", \"switchportMode\": \"trunk\",
\"shutdown\": \"false\"}"

response = session.post(url, data=payload)

print(response.text)
```

2. Get port-profile



```
import xmltodict
import requests
import json


url = 'http://192.168.56.3/api/port-profile'

headers = {'content-type':'application/json'}

session = requests.Session()
session.auth = ("admin","Wipro123")
get_vlans = session.get(url,headers=headers)
get_vlans_json = json.dumps(xmltodict.parse(get_vlans.text))
#get_vlans_json = json.loads(get_vlans_xml)
print(get_vlans_json)
```

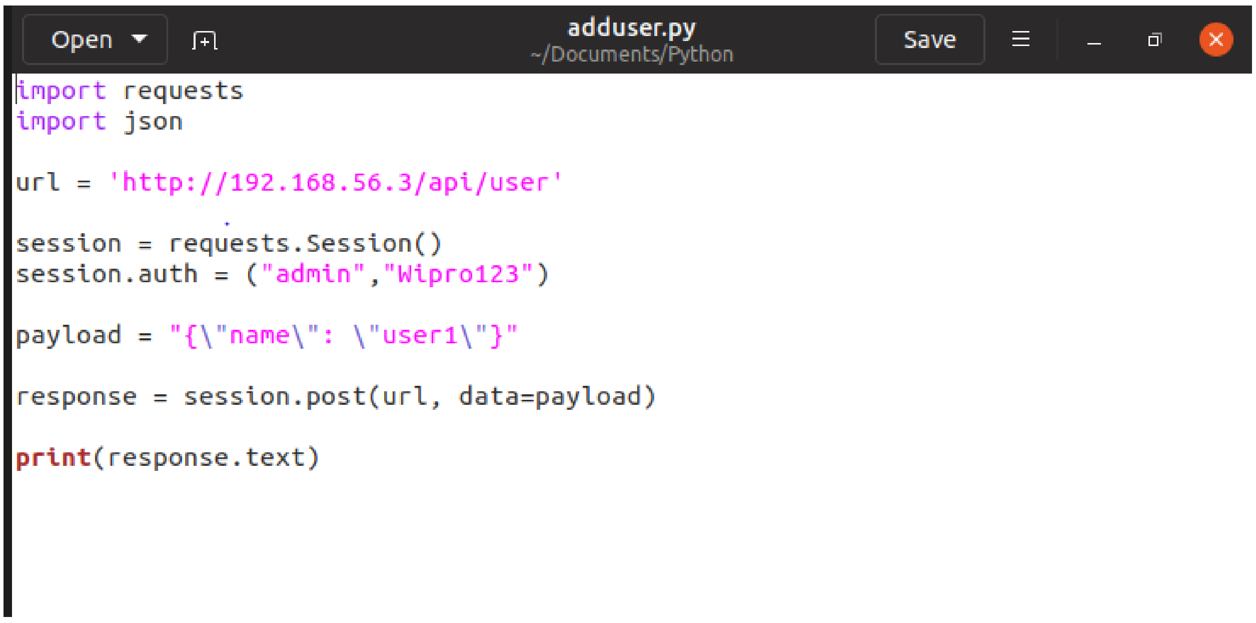
3. Delete port-profile



```
import requests
import json

url = 'http://192.168.56.3/api/port-profile/profile4'

session = requests.Session()
session.auth = ("admin","Wipro123")

#payload = "{\"name\": \"profile4\", \"switchportMode\": \"access\",
\"shutdown\": \"false\"}"

response = session.delete(url)

print(response.text)
```

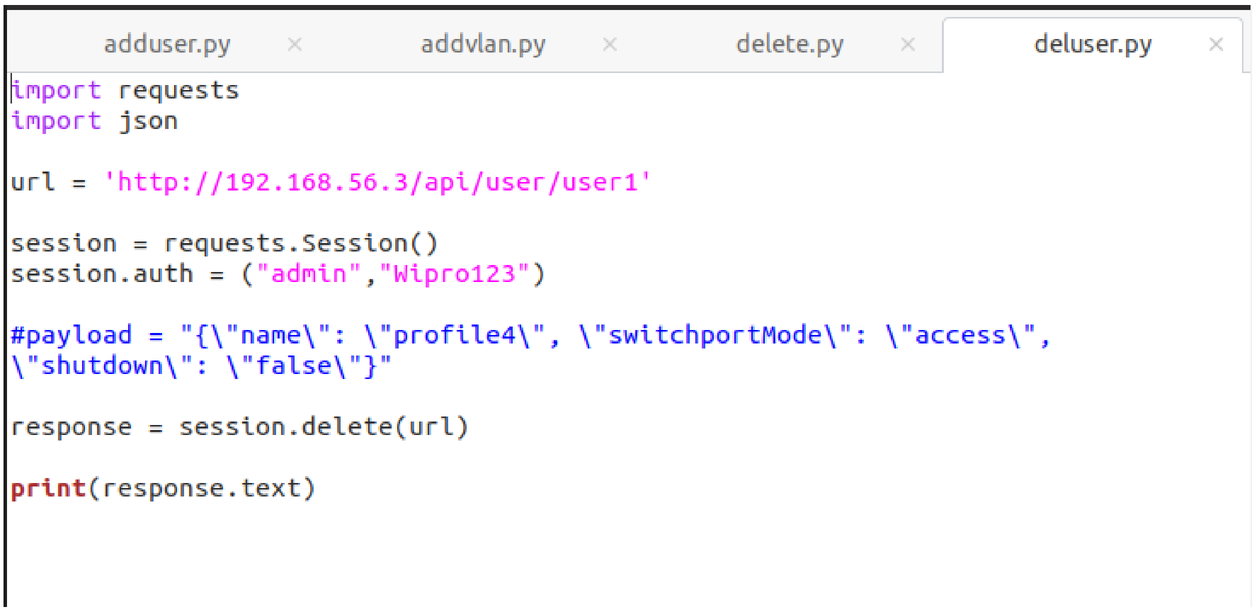
## 8.2 User

1. Create user



```
import requests
import json

url = 'http://192.168.56.3/api/user'

session = requests.Session()
session.auth = ("admin","Wipro123")

payload = "{\"name\": \"user1\"}"

response = session.post(url, data=payload)

print(response.text)
```

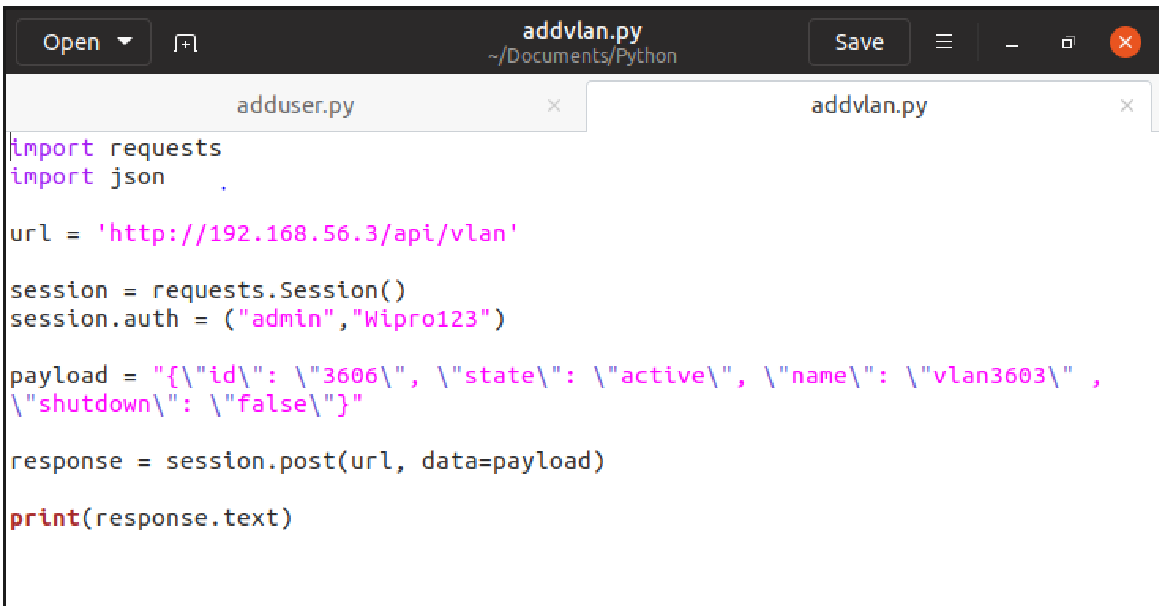
2. Delete user



```
import requests
import json

url = 'http://192.168.56.3/api/user/user1'

session = requests.Session()
session.auth = ("admin","Wipro123")

#payload = "{\"name\": \"profile4\", \"switchportMode\": \"access\",
\"shutdown\": \"false\"}"

response = session.delete(url)

print(response.text)
```

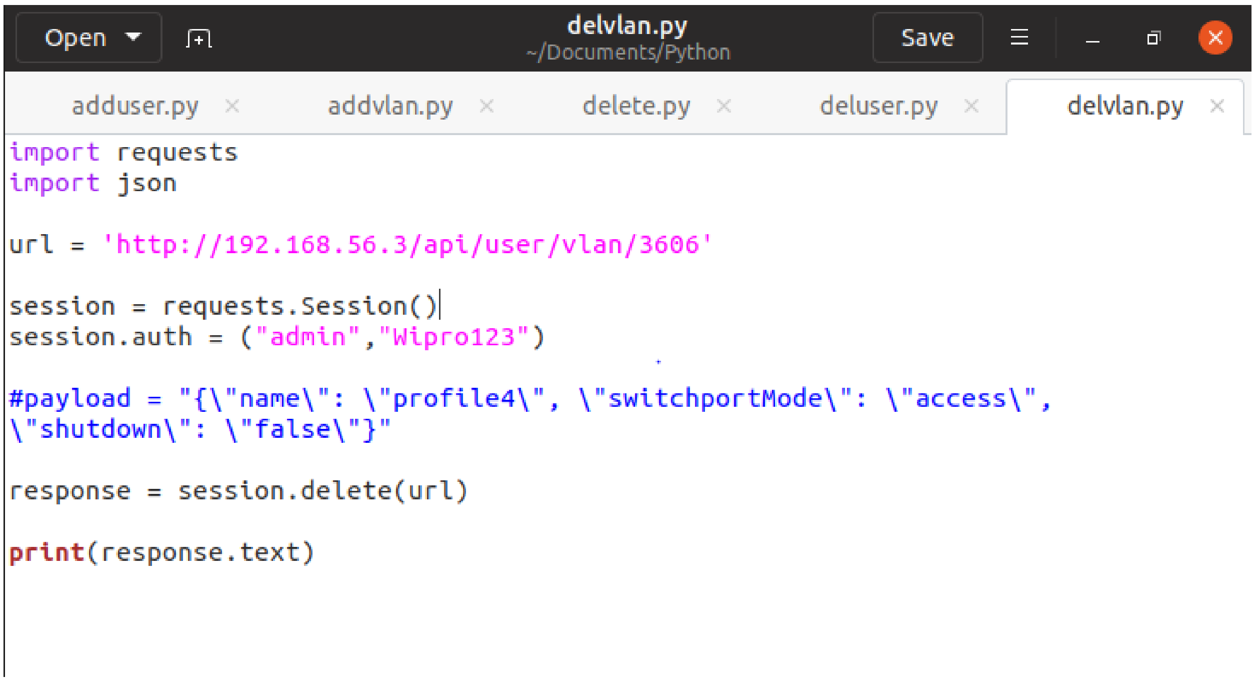
### 8.3 VLAN

1. Create VLAN



```
import requests
import json

url = 'http://192.168.56.3/api/vlan'

session = requests.Session()
session.auth = ("admin","Wipro123")

payload = "{\"id\": \"3606\", \"state\": \"active\", \"name\": \"vlan3603\" ,
\"shutdown\": \"false\"}"

response = session.post(url, data=payload)

print(response.text)
```

2. Delete VLAN



```
import requests
import json

url = 'http://192.168.56.3/api/user/vlan/3606'

session = requests.Session()
session.auth = ("admin","Wipro123")

#payload = "{\"name\": \"profile4\", \"switchportMode\": \"access\",
\"shutdown\": \"false\"}"

response = session.delete(url)

print(response.text)
```