pue IMPULSANDO EL CONOCIMIENTO TIC CUALIFICADO

EDUCACIÓN
CALIDAD Y CERTIFICACIÓN
EN SUS PROGRAMAS EDUCATIVOS

Programación de Redes – Becas Digitaliza - 2019
PUE – ITC – Formación de Instructores
Sesión 8 – APIC-EM Automatization & Python for Network Engineers

Iván Lago - Técnico Cisco Networking Academy ASC/ITC
PUE - ITC/ASC/CA

Index

Lab: APIC-EM, Postman & Python

APIC-EM

The Cisco Application Policy Infrastructure Controller Enterprise Module (APIC-EM) is an SDN controller. It is a software device that runs on servers and concentrates the control plane of the physical network infrastructure.

Cisco APIC-EM SDN controller communicates with the Physical Topology using standard Southbound API protocols such as SNMP, SSH and Telnet rather than a protocol like OpenFlow.

Easy abstraction of the network using a standard **REST API interface**. Therefore, it removes the need for the network staff to configure and manage every single networking device (routers, switches, access points, wireless controllers, etc.) one by one.

APIC-EM: log in

Virtualized APIC-EM Controllers are available in several DevNet Sandboxes:

Always On, NetAcad instances

- For NetAcad users only
- https://DevNetSBX-NetAcad-APICEM-3.cisco.com
  - **User:** *devnetuser* **PW:** Xj3BDqbU

Always on, public instance

- For to all DevNet users
- https://SandBoxAPICEM.cisco.com
  - **User**: *devnetuser* **PW**: *Cisco123!*

APIC-EM
Cisco Application Policy Infrastructure Controller Enterprise Module

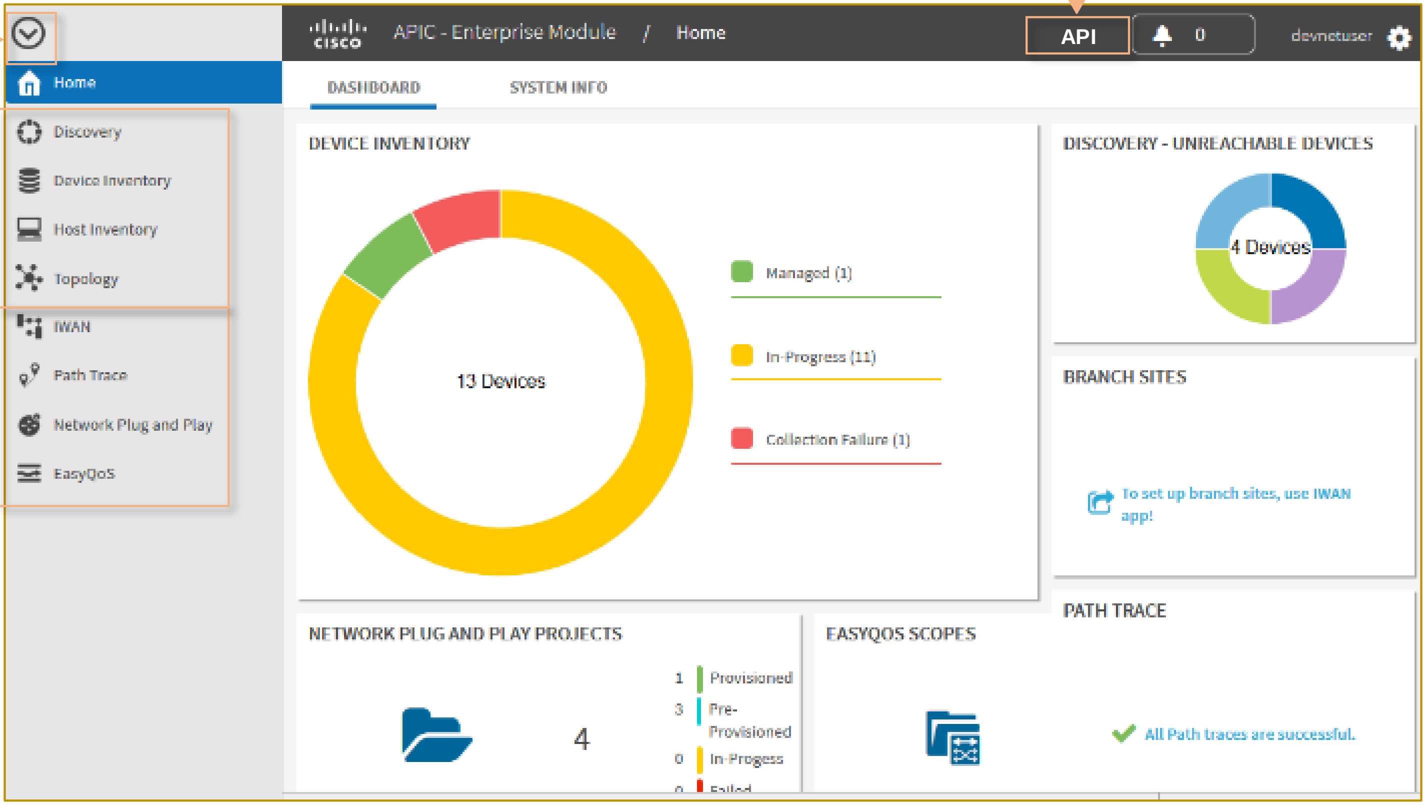
APIC-EM: home page (I)

API documentation

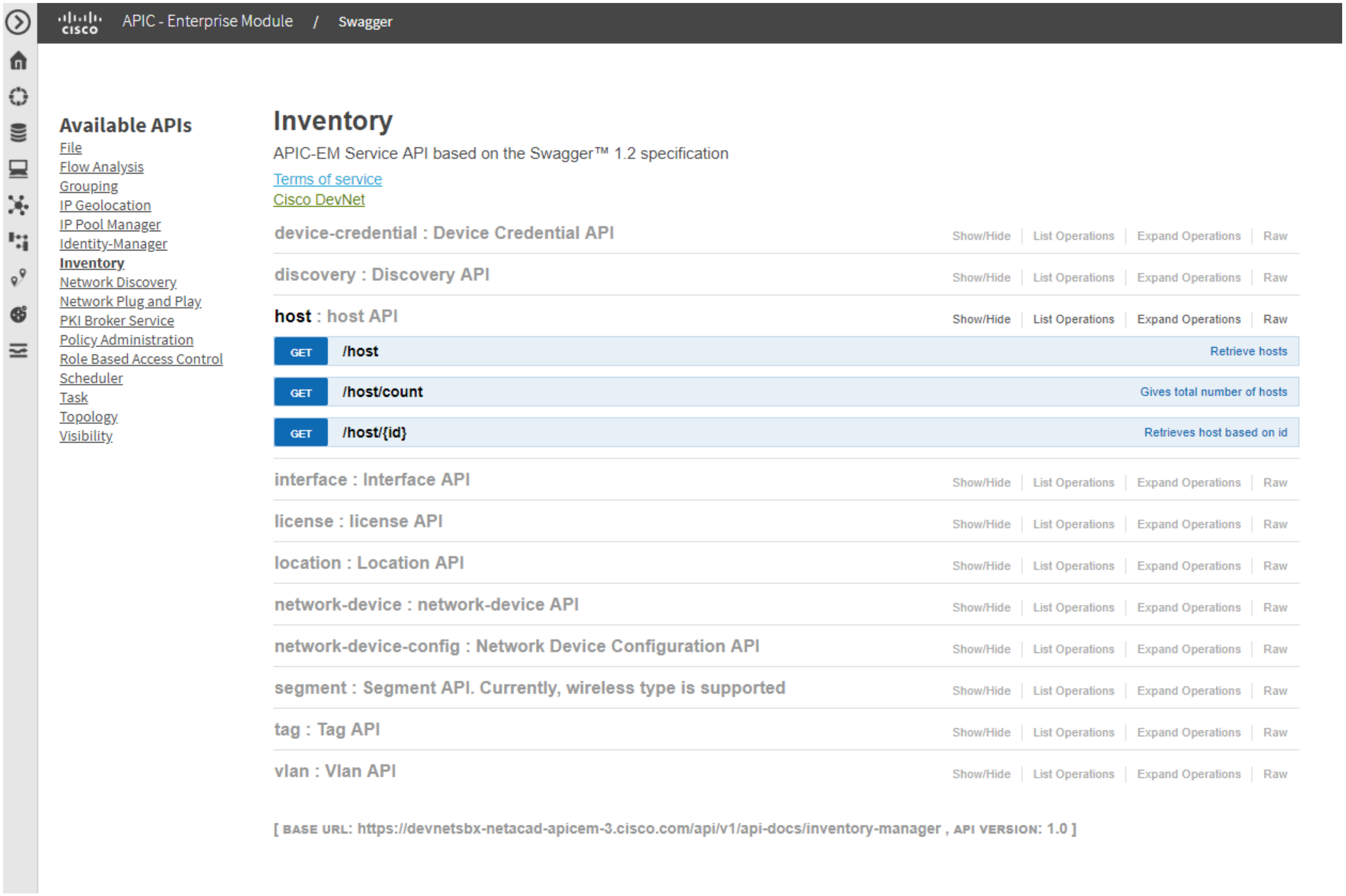
Expand Navigation Bar

Services

Applications



APIC-EM: API (I)



APIC - Enterprise Module / Swagger

Available APIs

- File
- Flow Analysis
- Grouping
- IP Geolocation
- IP Pool Manager
- Identity-Manager
- **Inventory**
- Network Discovery
- Network Plug and Play
- PKI Broker Service
- Policy Administration
- Role Based Access Control
- Scheduler
- Task
- Topology
- Visibility

Inventory

APIC-EM Service API based on the Swagger™ 1.2 specification

Terms of service

Cisco DevNet

device-credential : Device Credential API — Show/Hide | List Operations | Expand Operations | Raw

discovery : Discovery API — Show/Hide | List Operations | Expand Operations | Raw

host : host API — Show/Hide | List Operations | Expand Operations | Raw

GET	/host	Retrieve hosts
GET	/host/count	Gives total number of hosts
GET	/host/{id}	Retrieves host based on id

interface : Interface API — Show/Hide | List Operations | Expand Operations | Raw

license : license API — Show/Hide | List Operations | Expand Operations | Raw

location : Location API — Show/Hide | List Operations | Expand Operations | Raw

network-device : network-device API — Show/Hide | List Operations | Expand Operations | Raw

network-device-config : Network Device Configuration API — Show/Hide | List Operations | Expand Operations | Raw

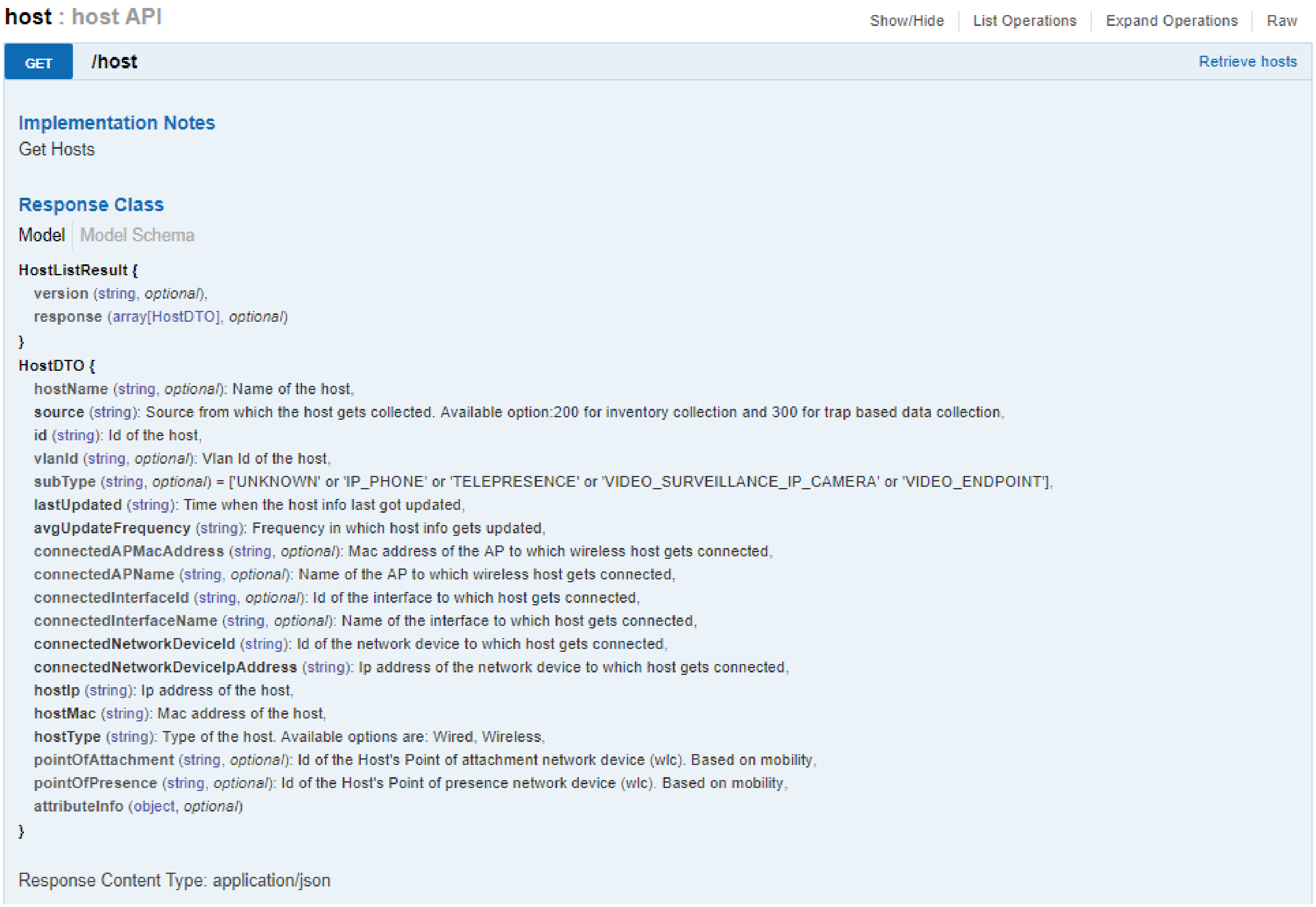
segment : Segment API. Currently, wireless type is supported — Show/Hide | List Operations | Expand Operations | Raw

tag : Tag API — Show/Hide | List Operations | Expand Operations | Raw

vlan : Vlan API — Show/Hide | List Operations | Expand Operations | Raw

[BASE URL: https://devnetsbx-netacad-apicem-3.cisco.com/api/v1/api-docs/inventory-manager , API VERSION: 1.0]

APIC-EM: API (II)



Implementation Notes

Get Hosts

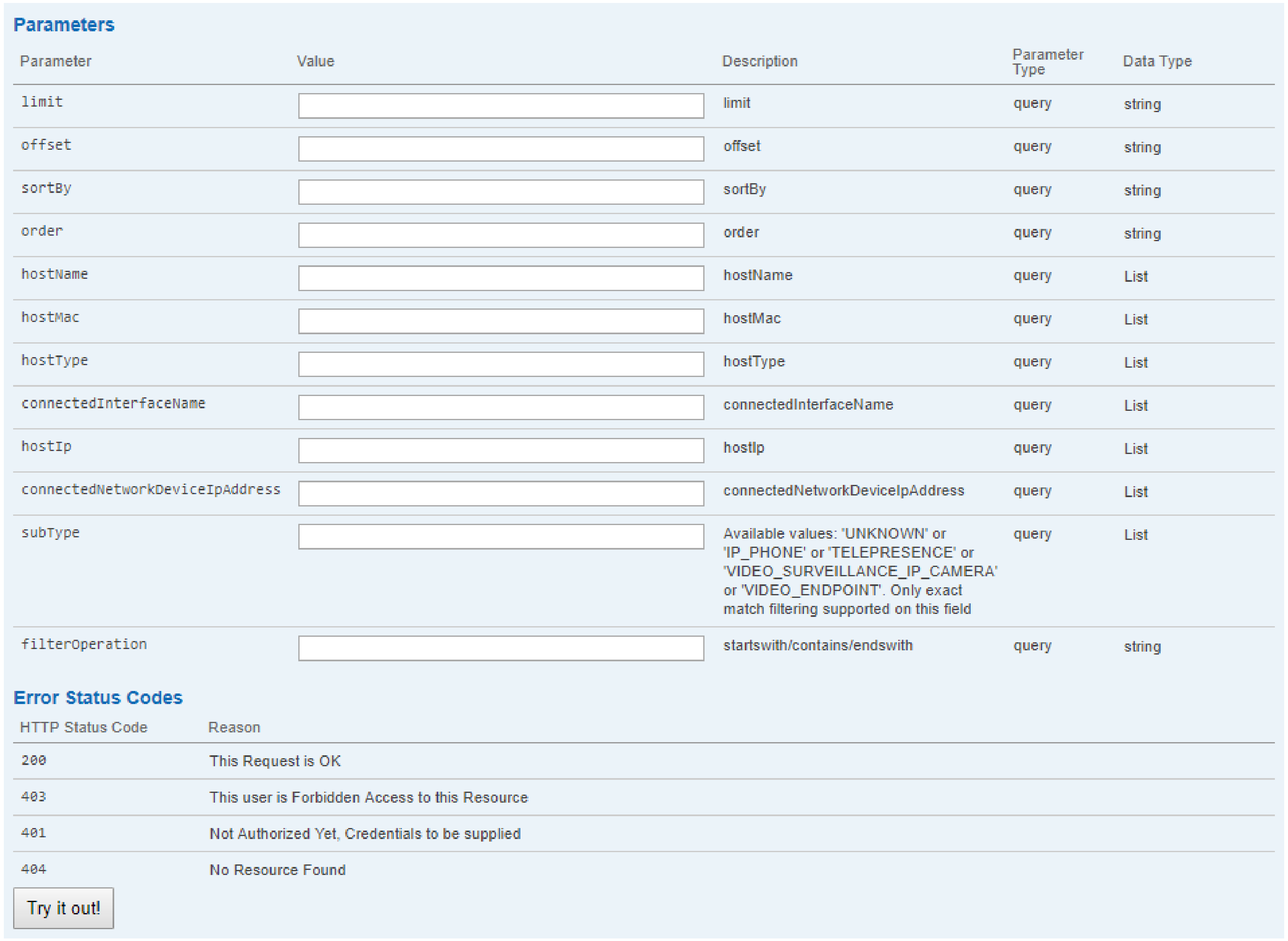
Response Class

Model | Model Schema

```
HostListResult {
   version (string, optional),
   response (array[HostDTO], optional)
}
HostDTO {
   hostName (string, optional): Name of the host,
   source (string): Source from which the host gets collected. Available option:200 for inventory collection and 300 for trap based data collection,
   id (string): Id of the host,
   vlanId (string, optional): Vlan Id of the host,
   subType (string, optional) = ['UNKNOWN' or 'IP_PHONE' or 'TELEPRESENCE' or 'VIDEO_SURVEILLANCE_IP_CAMERA' or 'VIDEO_ENDPOINT'],
   lastUpdated (string): Time when the host info last got updated,
   avgUpdateFrequency (string): Frequency in which host info gets updated,
   connectedAPMacAddress (string, optional): Mac address of the AP to which wireless host gets connected,
   connectedAPName (string, optional): Name of the AP to which wireless host gets connected,
   connectedInterfaceId (string, optional): Id of the interface to which host gets connected,
   connectedInterfaceName (string, optional): Name of the interface to which host gets connected,
   connectedNetworkDeviceId (string): Id of the network device to which host gets connected,
   connectedNetworkDeviceIpAddress (string): Ip address of the network device to which host gets connected,
   hostIp (string): Ip address of the host,
   hostMac (string): Mac address of the host,
   hostType (string): Type of the host. Available options are: Wired, Wireless,
   pointOfAttachment (string, optional): Id of the Host's Point of attachment network device (wlc). Based on mobility,
   pointOfPresence (string, optional): Id of the Host's Point of presence network device (wlc). Based on mobility,
   attributeInfo (object, optional)
}
```

Response Content Type: application/json

APIC-EM: API (III)



Parameters

Parameter	Value	Description	Parameter Type	Data Type
`limit`		limit	query	string
`offset`		offset	query	string
`sortBy`		sortBy	query	string
`order`		order	query	string
`hostName`		hostName	query	List
`hostMac`		hostMac	query	List
`hostType`		hostType	query	List
`connectedInterfaceName`		connectedInterfaceName	query	List
`hostIp`		hostIp	query	List
`connectedNetworkDeviceIpAddress`		connectedNetworkDeviceIpAddress	query	List
`subType`		Available values: 'UNKNOWN' or 'IP_PHONE' or 'TELEPRESENCE' or 'VIDEO_SURVEILLANCE_IP_CAMERA' or 'VIDEO_ENDPOINT'. Only exact match filtering supported on this field	query	List
`filterOperation`		startswith/contains/endswith	query	string

Error Status Codes

HTTP Status Code	Reason
200	This Request is OK
403	This user is Forbidden Access to this Resource
401	Not Authorized Yet, Credentials to be supplied
404	No Resource Found

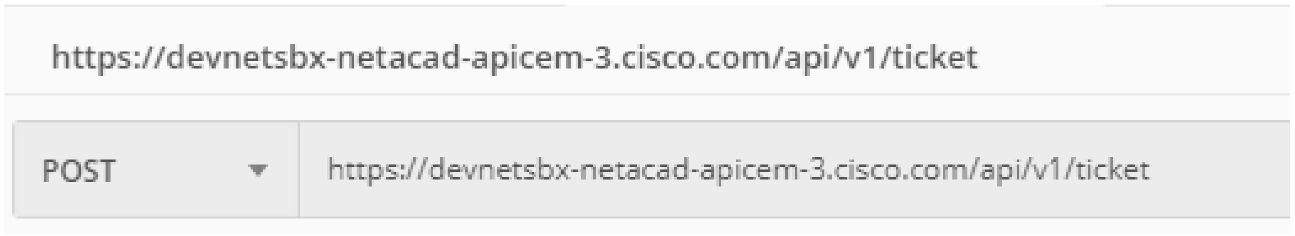
Postman

- https://www.getpostman.com/downloads/
- Available for Windows, Linux and macOS.
- After install it:
  - File -> Settings -> General -> SSL certificate verification: OFF

APIC-EM: getting service ticket

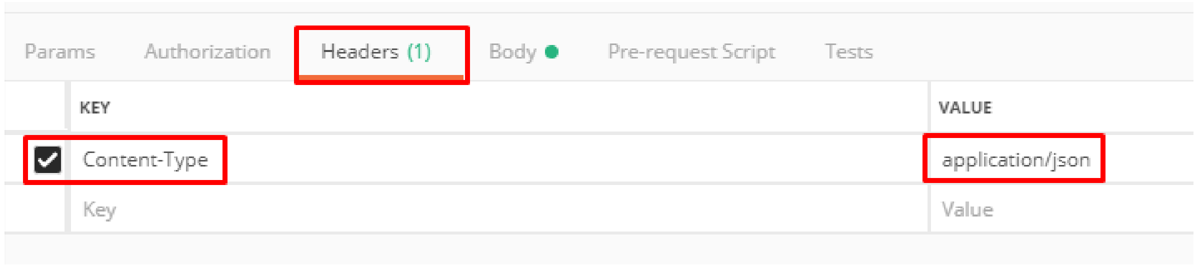
- The service ticket is a token to realize queries to get info (hosts, networks…)
- https://DevNetSBX-NetAcad-APICEM-3.cisco.com
  - **User:** *devnetuser* **PW:** Xj3BDqbU
- API -> Role Based Access Control -> ticket
  - POST /ticket
  - Model Schema -> click over yellow box
  - Value -> fill “password” and “username” with *Xj3BDqbU/devnetuser*
  - Try it out!
  - Response Body -> Service Ticket

Postman: getting service ticket

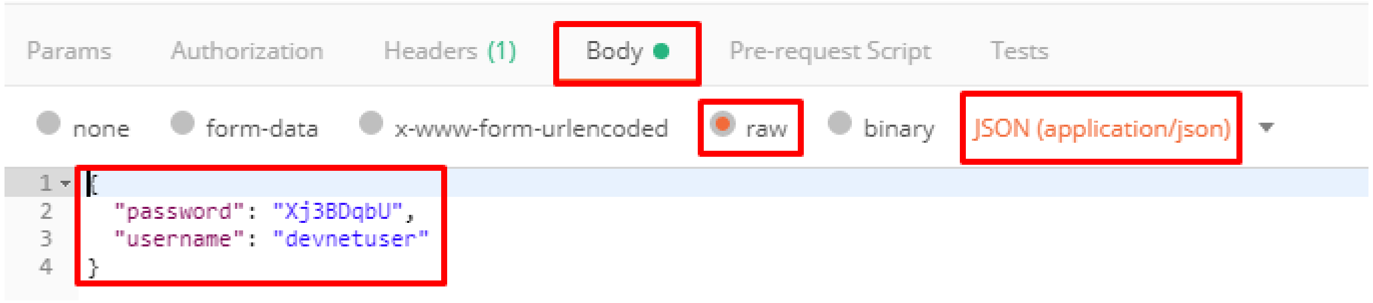


https://devnetsbx-netacad-apicem-3.cisco.com/api/v1/ticket

POST https://devnetsbx-netacad-apicem-3.cisco.com/api/v1/ticket



KEY	VALUE
✓ Content-Type	application/json
Key	Value

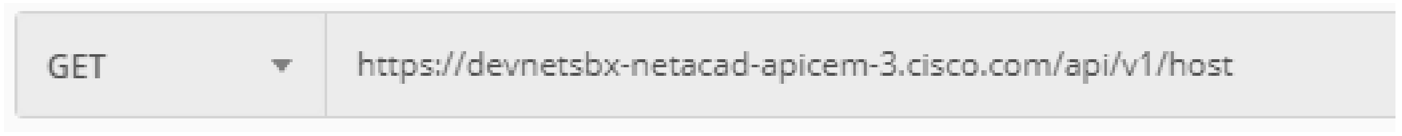


[✓ raw] JSON (application/json)

```
{
  "password": "Xj3BDqbU",
  "username": "devnetuser"
}
```

Postman: Getting Host Inventory (I)

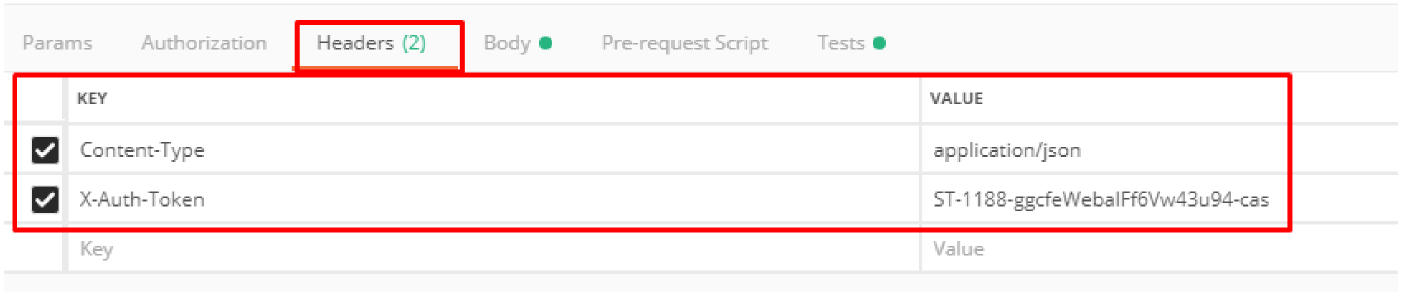
- Select method and URL:
  - Method: GET
  - URL: https://devnetsbx-netacad-apicem-3.cisco.com/api/v1/host



GET

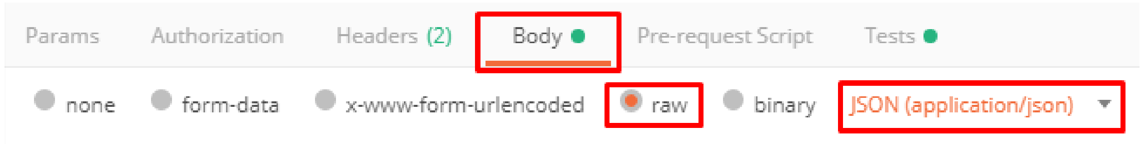
https://devnetsbx-netacad-apicem-3.cisco.com/api/v1/host

- Add Headers as key/value pairs:
  - Content-Type: application/json
  - X-Auth-Token: <id from Service Ticket>

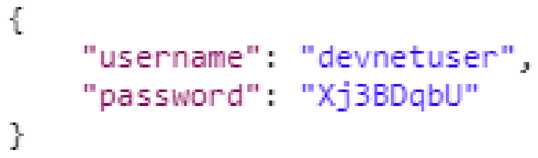


	KEY	VALUE
✓	Content-Type	application/json
✓	X-Auth-Token	ST-1188-ggcfeWebalFf6Vw43u94-cas
	Key	Value

Postman: Getting Host Inventory (II)



- Configure Body:
  - Choose JSON (application/json)
  - Select *“raw”*
  - Write the credentials:
    - *username": "devnetuser“*
    - *”password": "Xj3BDqbU"*



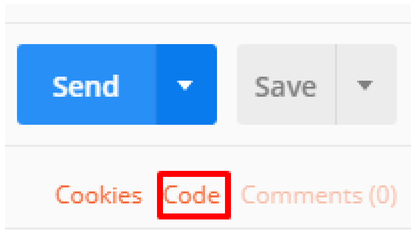
```
{
    "username": "devnetuser",
    "password": "Xj3BDqbU"
}
```

- Click over *Send*

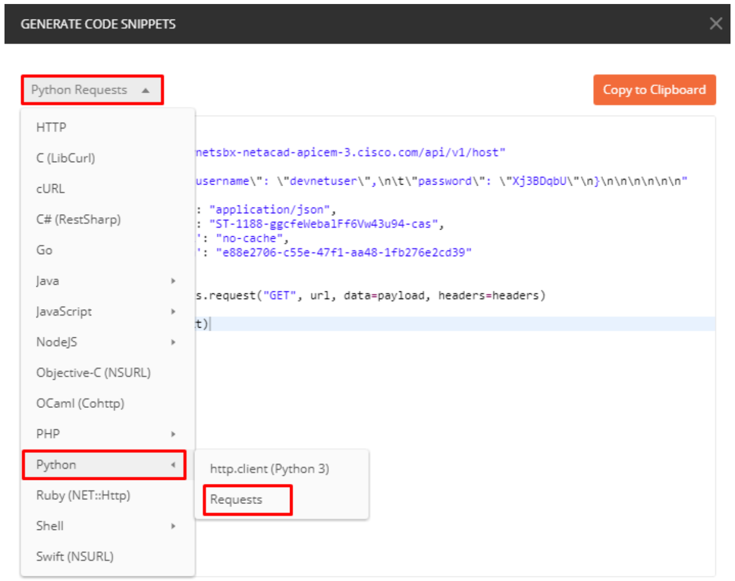
- Compare between JSON and raw or other formats

What happened with Python?

- Click over code



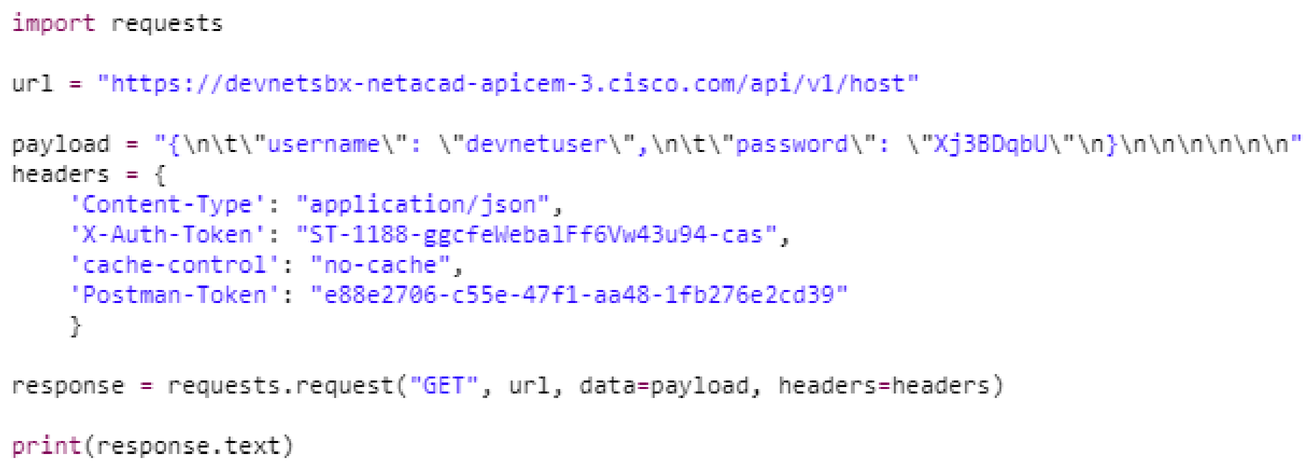
- Choose Python Requests



```
netsbx-netacad-apicem-3.cisco.com/api/v1/host"

username\": \"devnetuser\",\n\t\"password\": \"Xj3BDqbU\"\n}\n\n\n\n\n\n"

: "application/json",
: "ST-1188-ggcfeWebalFf6Vw43u94-cas",
': "no-cache",
': "e88e2706-c55e-47f1-aa48-1fb276e2cd39"


s.request("GET", url, data=payload, headers=headers)

t)
```

Python can do the same work that we did

- Postman gives us the code used by it, so it is possible

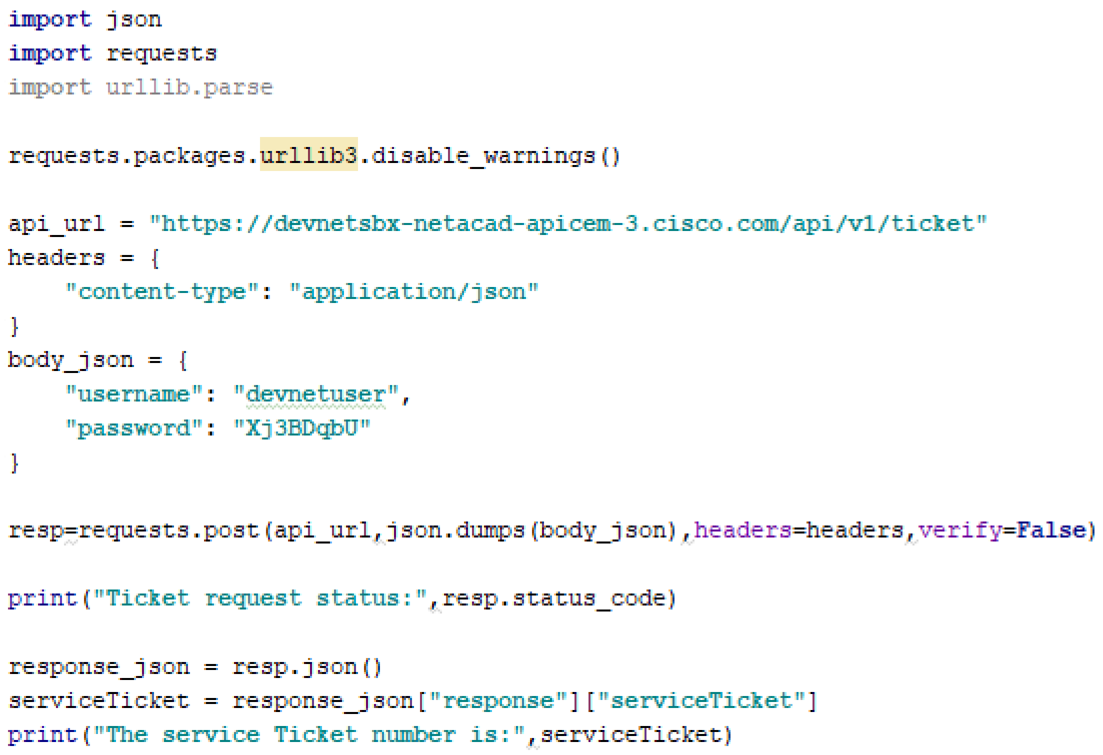


```python
import requests

url = "https://devnetsbx-netacad-apicem-3.cisco.com/api/v1/host"

payload = "{\n\t\"username\": \"devnetuser\",\n\t\"password\": \"Xj3BDqbU\"\n}\n\n\n\n\n\n"
headers = {
    'Content-Type': "application/json",
    'X-Auth-Token': "ST-1188-ggcfeWebalFf6Vw43u94-cas",
    'cache-control': "no-cache",
    'Postman-Token': "e88e2706-c55e-47f1-aa48-1fb276e2cd39"
    }

response = requests.request("GET", url, data=payload, headers=headers)

print(response.text)
```

- Let’s do it in our own!

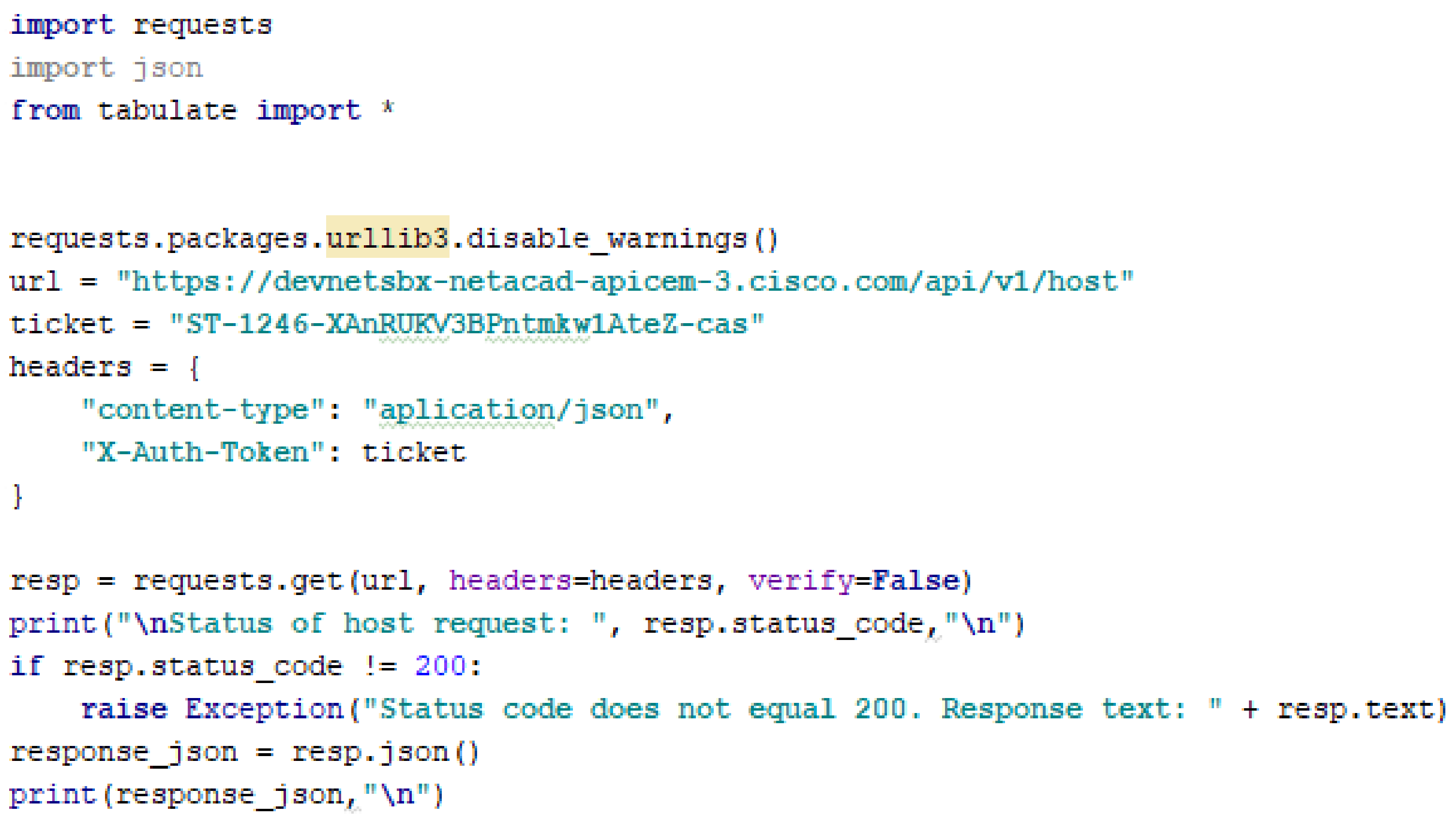
Getting a ticket with Python

- Disable_warnigs
  - Avoid error messages, such as certificate verifications
- json.dumps()
  - Create a json
- resp.json()
  - Json data to dict



```
import json
import requests
import urllib.parse

requests.packages.urllib3.disable_warnings()

api_url = "https://devnetsbx-netacad-apicem-3.cisco.com/api/v1/ticket"
headers = {
    "content-type": "application/json"
}
body_json = {
    "username": "devnetuser",
    "password": "Xj3BDqbU"
}

resp=requests.post(api_url,json.dumps(body_json),headers=headers,verify=False)

print("Ticket request status:",resp.status_code)

response_json = resp.json()
serviceTicket = response_json["response"]["serviceTicket"]
print("The service Ticket number is:",serviceTicket)
```

Getting Host Inventory with Python



```
import requests
import json
from tabulate import *


requests.packages.urllib3.disable_warnings()
url = "https://devnetsbx-netacad-apicem-3.cisco.com/api/v1/host"
ticket = "ST-1246-XAnRUKV3BPntmkw1AteZ-cas"
headers = {
    "content-type": "aplication/json",
    "X-Auth-Token": ticket
}

resp = requests.get(url, headers=headers, verify=False)
print("\nStatus of host request: ", resp.status_code,"\n")
if resp.status_code != 200:
    raise Exception("Status code does not equal 200. Response text: " + resp.text)
response_json = resp.json()
print(response_json,"\n")
```

Working & Parsing data

```
host_list = []
i = 0
for item in response_json["response"]:
    i+=1
    host = [
                i,
                item["hostType"],
                item["hostIp"],
                item["hostMac"]
            ]
    host_list.append( host )
table_header = ["Number", "Type", "IP", "MAC"]

print(tabulate(host_list, table_header))
```

Python Libraries

Libraries

Many ways to deal with libraries:

- Write ourselves
- Included in Python
- From PyPI (Python Package Index)
  - *pip install <library>*
- Download and install manually

Virtual Environments (I)

- Fully isolated functional environments on a single workstations (they may even run different python versions, different libraries...)

- Windows Set-up:
    - *pip install virtualenv*
    - *python3 -m venv name* --> create the environment
    - *name/Scripts/activate* --> activate virtual environment
    - *deactivate* --> deactivate virtual environment

Virtual Environments (II)

- Linux set-up:
  - *sudo apt install build-essential libssl-dev libffi-dev python3-dev*
  - *sudo apt install -y python3-venv* --> install environment
  - *pip install virtualenv*
  - *python3 -m venv name* --> create the environment
  - *source name/bin/activate* --> activate virtual environment
  - *Deactivate* --> deactivate virtual environment

Foundational libraries (I)

- Pretty Print
  - *from pprint import pprint*
  - *pprint(json_dict)*

- Python Interpreter Utilities
  - *import sys -->* allow you to use system functions
  - Example*: python3 -i test.py "prueba" "hello"*
    - *sys.argv[1] --> 'prueba'*
    - *sys.argv[2] --> 'hello'*
    - *sys.exit("DAMM") --> 'damm' and exits from .py*

Foundational libraries (II)

- Operating System Interfaces
  - *import os*
  - *os.getcwd()* --> know working directory
  - *os.chdir("../")* --> change working directory
  - *os.environ["USER"]* --> print the name of the user
  - *os.environ["VARIABLE_YOU_WANT"] = "something to put into"* --> create a variable
  - *os.environ["VARIABLE_YOU_WANT"]*

- Date and Time Utilities
  - *import datetime*
  - *right_now = datetime.datetime.now()*
  - *four_weeks_from_now = right_now+datetime.timedelta(weeks=4)*
  - *date_display_format = "%I:%m %p on %B %w, %Y“*
  - *four_weeks_from_now.strftime(date_display_format)*

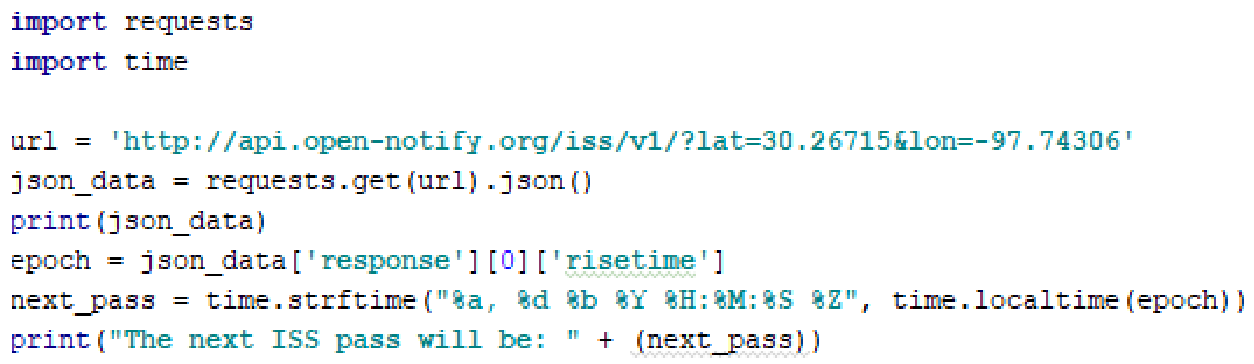
Data Formating:
JSON, XML, YAML & CSV

Libraries to work with Data (I)

XML --> we have xmltodict

- *pip install xmltodict*
- *import xmltodict*
- *xml_example=open("xml_example.xml").read()*
- *xml_dict = xmltodict.parse(xml_example)*
- *print(xml_dict["interface"]["ipv4"]["address"]["ip"]) ->* search values as a common dict
  - we can also unparse the data --> *xmltodict.unparse(xml_dict)*

JSON -->



```
import requests
import time

url = 'http://api.open-notify.org/iss/v1/?lat=30.26715&lon=-97.74306'
json_data = requests.get(url).json()
print(json_data)
epoch = json_data['response'][0]['risetime']
next_pass = time.strftime("%a, %d %b %Y %H:%M:%S %Z", time.localtime(epoch))
print("The next ISS pass will be: " + (next_pass))
```

Libraries to work with Data (II)

YAML --> we have PyYAML

- *pip install PyYAML*
- *import yaml*
- *from pprint import pprint*
- *import os*
- *print(os.getcwd())*

- Example:
  - *open(“<path>/yaml_example.yaml").read()*
  - *pprint(yml_example)*
  - *yaml_python = yaml.load(yml_example)* -> convert to dict pprint(yaml_python)
  - *print(yaml_python["ietf-interfaces:interface"]["ietf-ip:ipv4"]["address"][0]["ip"])*
  - *yaml.dump(yaml_python)* --> come back to yaml

Libraries to work with Data (III)

Example of CSV data:

- *import csv*
- *from pprint import pprint*
- *csv_example = open(“path/csv_example.csv")*
- *csv_python = csv.reader(csv_example)*
- *for row in csv_python:*
  - *print("{} is in {} and has IP {}".format(row[0], row[2], row[1]))*

Gracias por vuestra atención

pue

IMPULSANDO EL CONOCIMIENTO
TIC CUALIFICADO

Iván Lago - Técnico Cisco Networking Academy ASC/ITC
PUE - ITC/ASC/CA
Área de Proyectos de Educación