

# Cisco Networking Academy

Innovate like a Technologist. Think like an Entrepreneur. Act as a Social Agent.





# Network Programmability with Cisco APIC-EM

## Chapter 2: Programming the APIC-EM

Chotipat Pornavalai, chotipat@it.kmitl.ac.th

Faculty of Information Technology, KMITL

25 May 2018



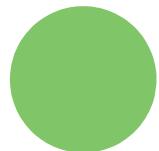
# Introduction

# Welcome

# Agenda



Network Programmability



Programming the APIC-EM REST Application  
Programming Interface (API)



Putting it All Together: A Simple Python APIC-  
EM Application

# Workshop Objectives

**At the end of this workshop you will be able to:**

- Explain how the Cisco APIC-EM enhances network management and performance with software defined networking (SDN) and network programmability.
- Create an inventory of network devices by using the APIC-EM REST API.
- Create Python software tools for working with the APIC-EM API.

**Please note:** You are NOT expected become software developers or network programmers - yet!

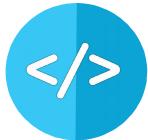
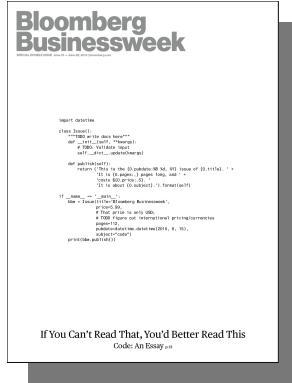
# Computer Requirements

- Python and IDLE
- Installed Python Modules:
  - requests
  - tabulate
- The Postman application

# Why are we here?



Everything becomes connected



Everything becomes software-based



Everything generates data



Everything can be automated



Everything needs to be secured

Networking

Programmability

Security

# Building an Industry Ecosystem with DevNet



Cisco's portfolio as a Platform for Innovation

<https://developer.cisco.com/>



IoT



Cloud



Networking



Data Center



Security



Analytics &  
Automation



Open Source

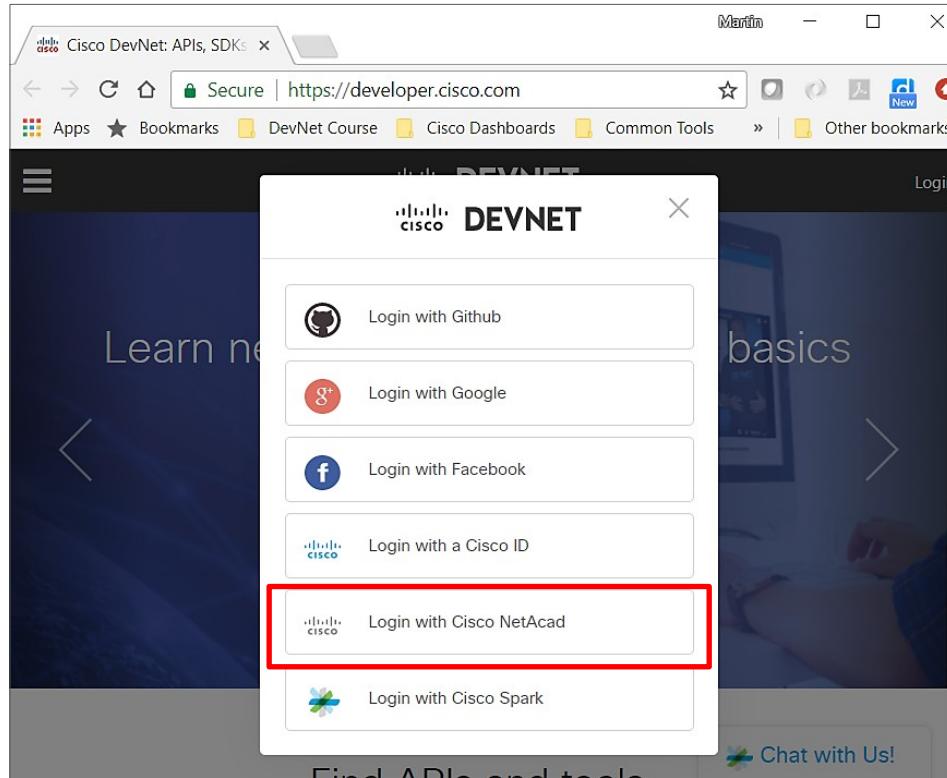


Collaboration



Mobility

# Login to Devnet





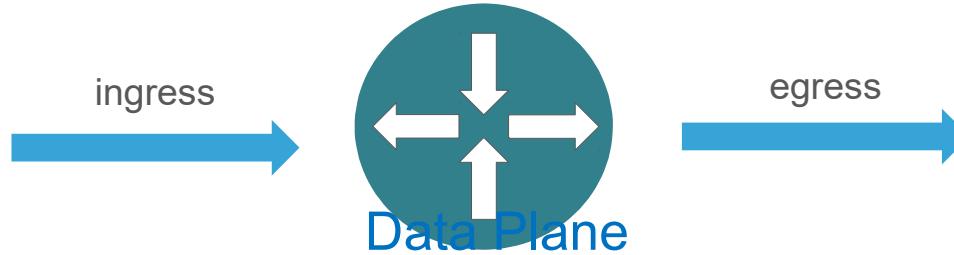
# Network Programmability

# Network Programmability Concepts

# SDN: Control Plane and Data Plane

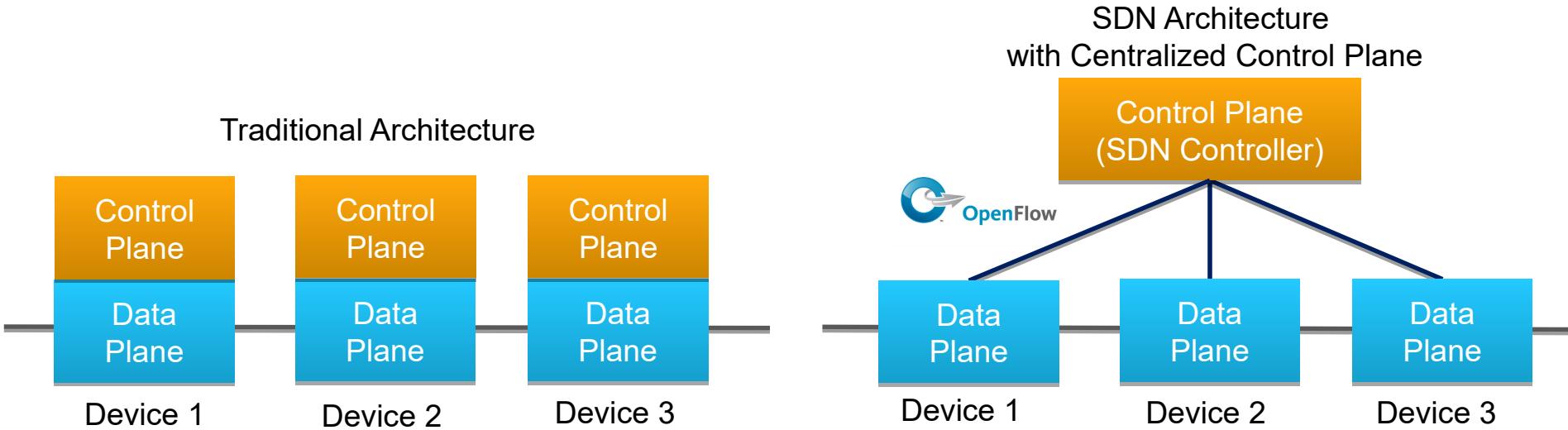
## Control Plane

Hardware	Purpose	Example Processes
Device CPU	makes decisions about where traffic is sent	routing protocols, spanning tree, AAA, SNMP, CLI



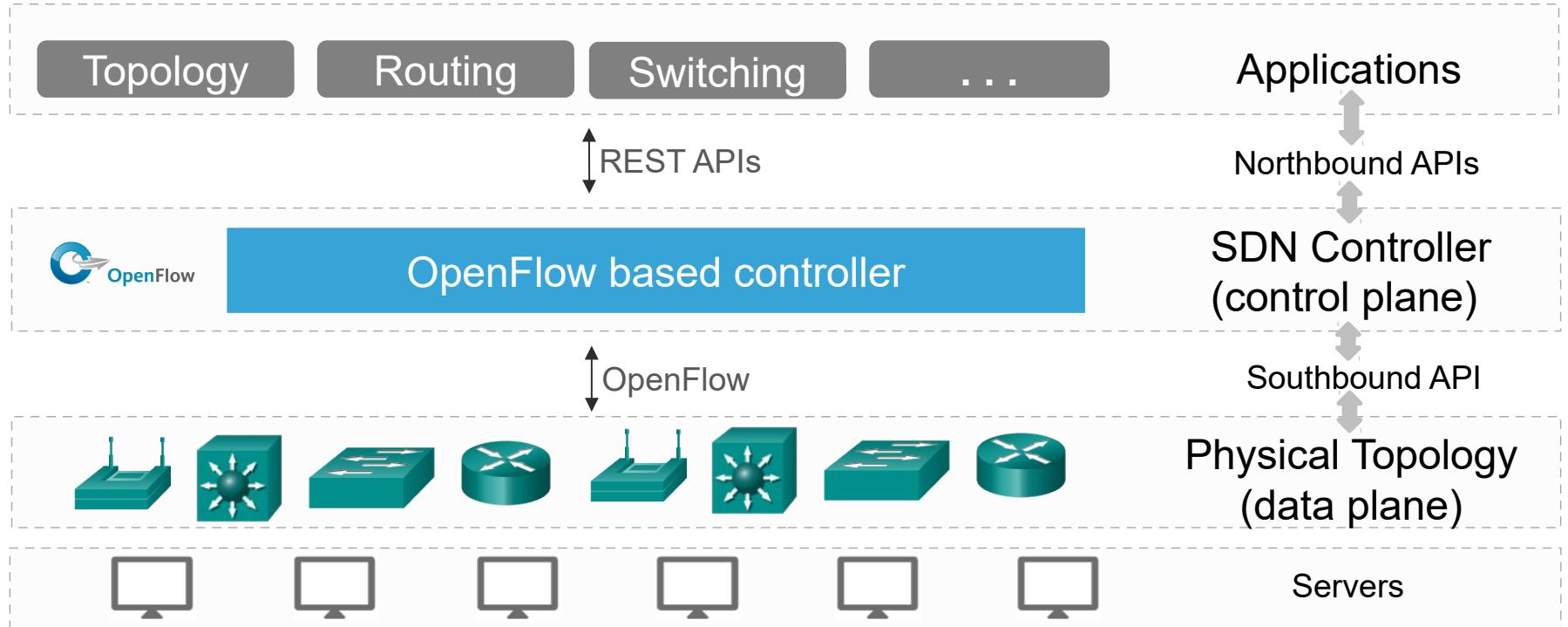
Hardware	Purpose	Example Processes
Dedicated ASICs	forwards traffic to the selected destination	packet switching, L2 switching, QoS, policies, ACLs

# Traditional and SDN Architectures

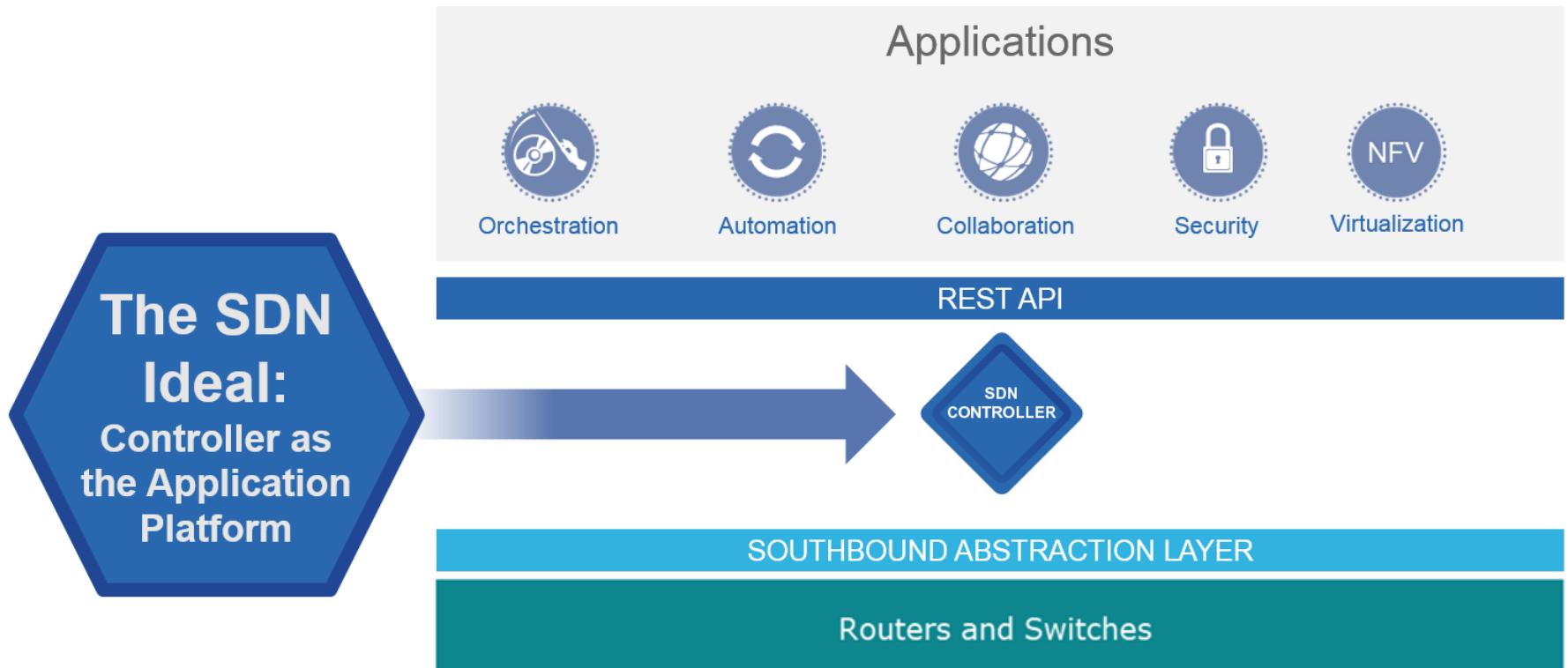


OpenFlow is a protocol between SDN controllers and network devices, as well as a specification of the logical structure of the network switch functions.

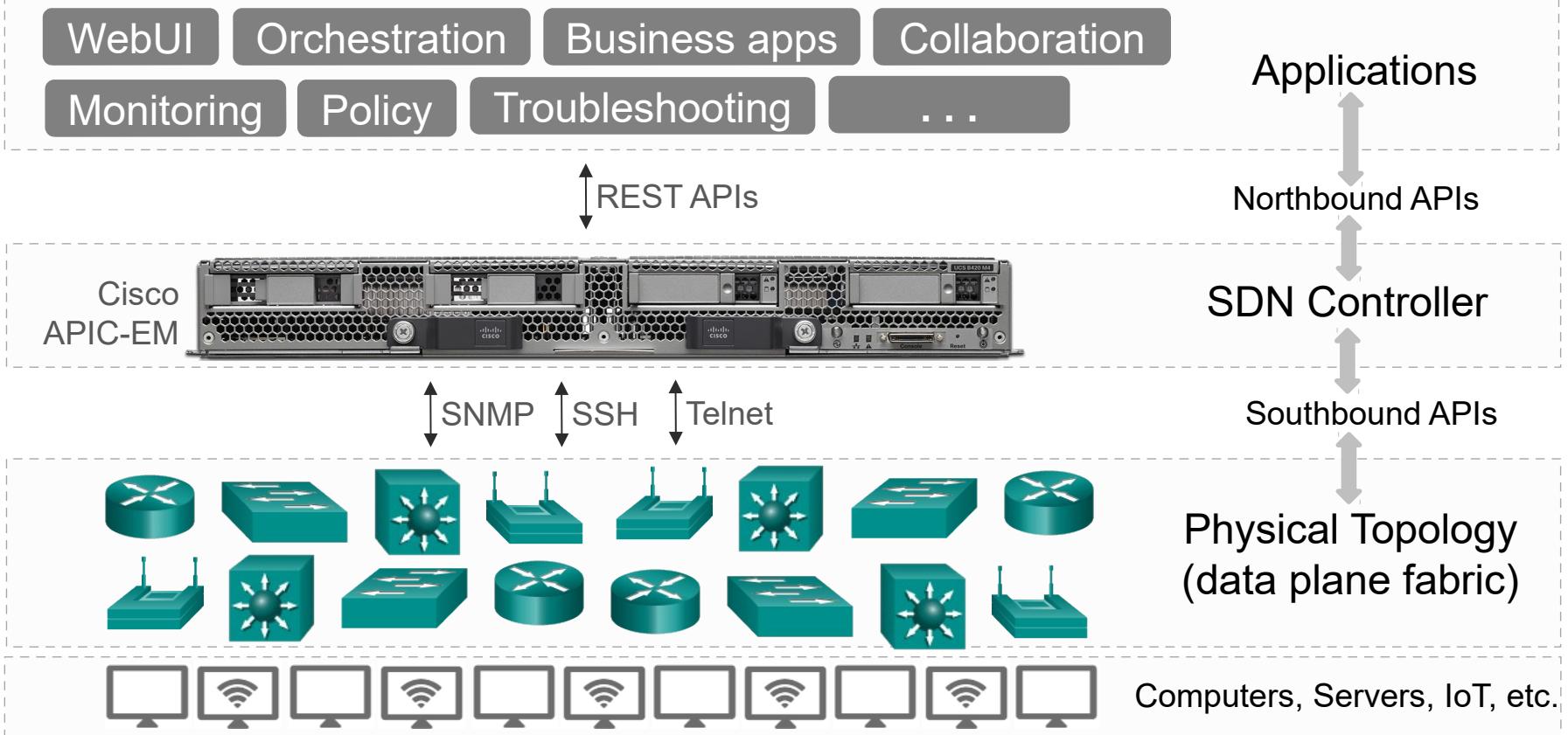
# OpenFlow SDN Model



# Network-Wide Abstractions Simplify the Network



# Cisco SDN Model with APIC-EM



# The APIC-EM

# What is the APIC-EM?



The Cisco Application Policy Infrastructure Controller Enterprise Module (APIC-EM):

- A Software-Defined Networking (SDN) controller for enterprise networks
- A virtual, software-only, or physical appliance
- Creates an intelligent, open, programmable network with open APIs
- Can transform business-intent policies into dynamic network configuration
- Provides a single point for network-wide automation and control

# 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-1.cisco.com>
- <https://DevNetSBX-NetAcad-APICEM-2.cisco.com>
- <https://DevNetSBX-NetAcad-APICEM-3.cisco.com>
- **User/PW:** Ask your instructor

Always on, public instance

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



# APIC-EM – Log in

A virtualized APIC-EM Controller is available in a DevNet Sandbox:

Always on, NetAcad only instances

- For NetAcad users in specific regions.
- <https://DevNetSBX-NetAcad-APICEM-1.cisco.com>
- <https://DevNetSBX-NetAcad-APICEM-2.cisco.com>
- <https://DevNetSBX-NetAcad-APICEM-3.cisco.com>
- **User/PW:** Ask your instructor

Always on, public instance

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



# APIC-EM Home Page

The screenshot shows the APIC-EM Home Page. A red box highlights the 'API' button in the top right corner. Another red box highlights the navigation bar icon (a downward arrow inside a square) at the top left. A third red box highlights the expanded navigation bar on the left side, which includes sections for Discovery, Device Inventory, Host Inventory, Topology, IWAN, Path Trace, Network Plug and Play, and EasyQoS.

Expand Navigation Bar

Services

Applications

API documentation

APIC - Enterprise Module / Home

DASHBOARD SYSTEM INFO

DEVICE INVENTORY

13 Devices

DISCOVERY - UNREACHABLE DEVICES

4 Devices

Managed (1)

In-Progress (11)

Collection Failure (1)

BRANCH SITES

To set up branch sites, use IWAN app!

PATH TRACE

NETWORK PLUG AND PLAY PROJECTS

4

EASYQOS SCOPES

1 Provisioned

3 Pre-Provisioned

0 In-Progress

0 Failed

All Path traces are successful.

# APIC-EM Applications

- **Network Plug-and-Play (PnP)**

Provides a unified approach to provision enterprise networks comprised of Cisco routers, switches, and wireless access points with a near-zero-touch deployment experience.

- **Easy QoS**

Provides a simple way to classify and assign application priority.

- **Intelligent WAN (IWAN)**

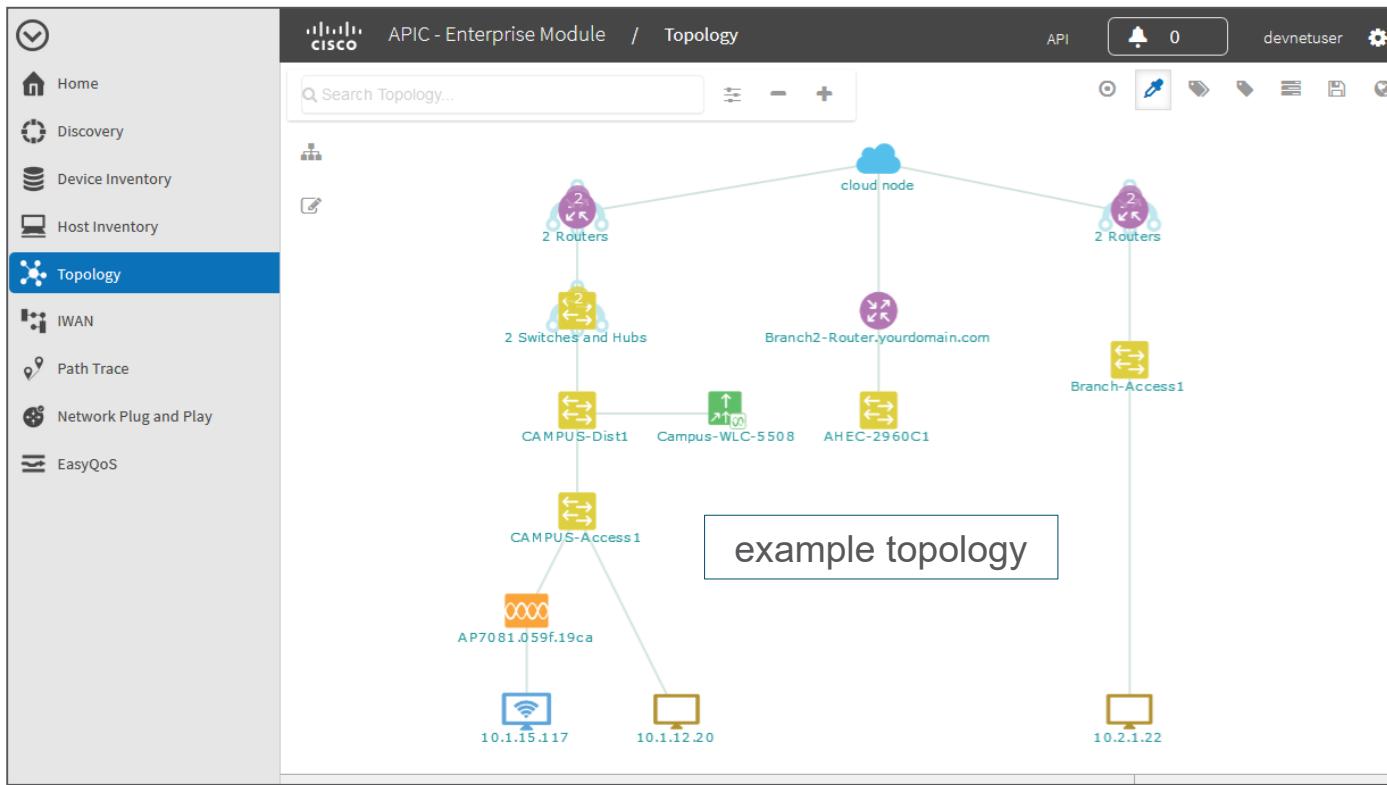
Simplifies WAN deployments by providing an intuitive, policy-based interface that helps IT abstract network complexity and design for business intent.

- **Path Trace**

Greatly eases and accelerates the task of connection monitoring and troubleshooting.

More [information](#) on the Cisco APIC-EM.

# APIC-EM Topology Page





# Programming the APIC-EM REST API

# REST APIs

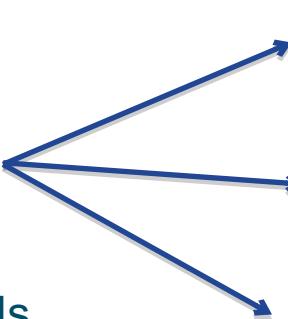
# What is so great about REST\*?



## Cisco APIC-EM REST APIs

- Hosts
- Devices
- Users
- + more

## How does this work?

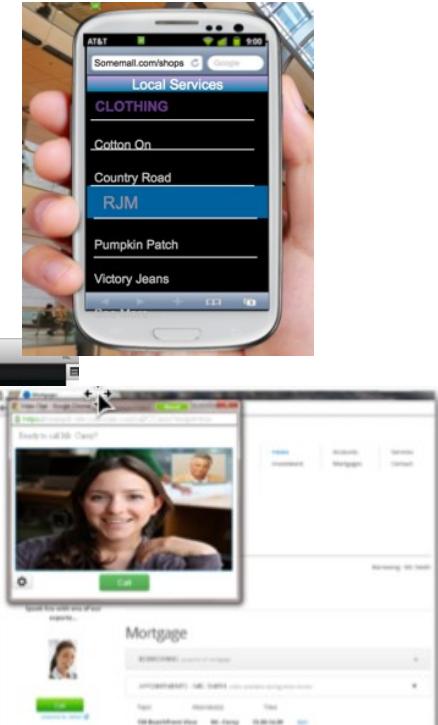


Easy to use:

- In mobile apps
- In console apps
- In web apps

```
Hosts=
192.168.68.130
26.6.6.9
26.6.6.11
26.6.6.10
25.5.5.56
14.4.4.12
14.4.4.17
14.4.4.11
14.4.4.10
12.2.2.11
12.2.2.10
12.2.2.12

Policies=
419abccf-c2c3-4421-a96a-f3de3981ce5f
100c8e40-fd50-4fcf-a099-cb0b0436329c0
2a9460ba-7db2-4790-b1b3-8d13166c69e5
fd4bcc11-c0a1-4d5f-8bdc-636ccb6f8fb3
2c27beaa-ddef-4b77-ada6-a378d392bbe3
```



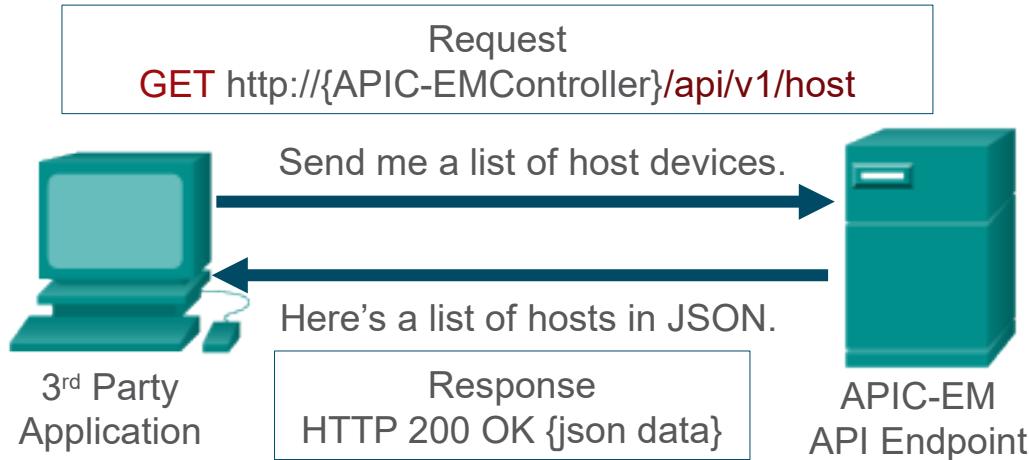
\*representational state transfer (REST).

# REST APIs

- Use HTTP protocol methods and transport
- API **endpoints** exist as server processes that are accessed through URIs
- Webpages present data and functionality in human-machine interaction driven by a user.
- APIs present data and functionality in machine-machine interactions driven by software.

Directory of Public APIs: <https://www.programmableweb.com/apis/directory>

# How does this work?



# Anatomy of a REST Request

REST requests require the following elements (requirements may differ depending on the API):

## **Method**

- GET (retrieve), POST (create), PUT (update), DELETE (remove)

## **URL**

- Example: `http://{APIC-EMController}/api/v1/host`

## **Authentication**

- Basic HTTP, OAuth, none, Custom

## **Custom Headers**

- HTTP Headers
- Example: Content-Type: application/json

## **Request Body**

- JSON or XML containing data needed to complete request

# What is in the Response?

## HTTP Status Codes

- <http://www.w3.org/Protocols/HTTP/HTRESP.html>
- 200 OK
- 201 Created
- 401, 403 Authorization error
- 404 Resource not found
- 500 Internal Error

## Headers

## Body

- JSON
- XML

```
1 {  
2     "version": "0.0",  
3     "response": "349117ce-3c7f-4e14-bc6f-83071e990198: Acl Policy  
Appended Successfully on the Device : 9cb0df12-b9f7-4551-932e-  
3391974da58f"  
4 }
```

Example output of a HTTP response in the Postman application

# JSON and XML

JSON

```
1  [{}]
2  {
3    "response": {
4      "request": {
5        "sourceIP": "10.1.15.117",
6        "destIP": "10.2.1.22",
7        "periodicRefresh": false,
8        "id": "feb8f5c6-56d1-45ec-9a49-bd4afac5c887",
9        "status": "COMPLETED",
10       "createTime": 1506693815419,
11       "lastUpdateTime": 1506693823127
12     },
13     "lastUpdate": "Fri Sep 29 14:03:43 UTC 2017",
14     "networkElementsInfo": [
15       {
16         "id": "48cdeb9b-b412-491e-a80c-7ec5bbe98167",
17         "type": "wireless",
18         "ip": "10.1.15.117",
19         "linkInformationSource": "Switched"
20       },
21       {
22         "id": "cd6d9b24-839b-4d58-adfe-3fdf781e1782",
23         "name": "AP7081.059f.19ca",
24         "type": "Unified AP",
25         "ip": "10.1.14.3",
26         "role": "ACCESS",
27         "linkInformationSource": "Switched",
28         "tunnels": [
29           "CAPWAP Tunnel"
30         ]
31       },
32       {
33         "id": "5b5ea8da-8c23-486a-b95e-7429684d25fc",
34         "name": "CAMPUS-Access1",
35         "type": "Switches and Hubs",
36         "ip": "10.1.12.1",
37         "ingressInterface": {
38           "physicalInterface": {
39             "id": "dd2c47ea-ad19-4a1e-ad0e-82d9deefd61b",
40             "name": "GigabitEthernet1/0/26"
41           }
42         },
43         "egressInterface": {
44           "physicalInterface": {
45             "id": "38c72319-855e-43bc-8458-94f695d435b6",
46             "name": "GigabitEthernet1/0/1"
47           }
48         }
49       }
50     ]
51   }
52 }
```

XML

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <response>
3   <request>
4     <sourceIP>10.1.15.117</sourceIP>
5     <destIP>10.2.1.22</destIP>
6     <periodicRefresh>false</periodicRefresh>
7     <id>feb8f5c6-56d1-45ec-9a49-bd4afac5c887</id>
8     <status>COMPLETED</status>
9     <createTime>1506693815419</createTime>
10    <lastUpdateTime>1506693823127</lastUpdateTime>
11  </request>
12  <lastUpdate>Fri Sep 29 14:03:43 UTC 2017</lastUpdate>
13  <networkElementsInfo>
14    <id>48cdeb9b-b412-491e-a80c-7ec5bbe98167</id>
15    <type>wireless</type>
16    <ip>10.1.15.117</ip>
17    <linkInformationSource>Switched</linkInformationSource>
18  </networkElementsInfo>
19  <networkElementsInfo>
20    <id>cd6d9b24-839b-4d58-adfe-3fdf781e1782</id>
21    <name>AP7081.059f.19ca</name>
22    <type>Unified AP</type>
23    <ip>10.1.14.3</ip>
24    <role>ACCESS</role>
25    <linkInformationSource>Switched</linkInformationSource>
26    <tunnels>CAPWAP Tunnel</tunnels>
27  </networkElementsInfo>
28  <networkElementsInfo>
29    <id>5b5ea8da-8c23-486a-b95e-7429684d25fc</id>
30    <name>CAMPUS-Access1</name>
31    <type>Switches and Hubs</type>
32    <ip>10.1.12.1</ip>
33    <ingressInterface>
34      <physicalInterface>
35        <id>dd2c47ea-ad19-4a1e-ad0e-82d9deefd61b</id>
36        <name>GigabitEthernet1/0/26</name>
37      </physicalInterface>
38    </ingressInterface>
39    <egressInterface>
40      <physicalInterface>
41        <id>38c72319-855e-43bc-8458-94f695d435b6</id>
42        <name>GigabitEthernet1/0/1</name>
43      </physicalInterface>
44    </egressInterface>
```

# What about authentication?

- **None:** The API resource is public and anybody can place the request.
  - **Basic HTTP:** The username and password are passed to the server in an encoded string.
  - **OAuth:** Open standard for HTTP authentication and session management. Creates an access token associated to a specific user that also specifies the user rights.
  - **Token:** A token is created and passed with each API call, but there is no session management and tracking of clients which simplifies interaction between the server and client.
- APIC-EM uses **Token** for authentication management. The APIC-EM calls this token a service ticket.

# View the APIC-EM Swagger Documentation

The screenshot shows the APIC-EM Swagger documentation interface. The left sidebar lists "Available APIs" including File, Flow Analysis, Grouping, IP Geolocation, IP Pool Manager, Identity-Manager, Inventory, Network Discovery, Network Plug and Play, PKI Broker Service, Policy Administration, and Role Based Access Control. The "Role Based Access Control" item is selected and highlighted in blue. The main content area displays the "Role Based Access Control" API documentation. It includes sections for aaa, role, ticket, and user. The ticket section is expanded, showing operations for /ticket, /ticket/attribute, /ticket/attribute/{attribute}, and /ticket/{ticket}. Each operation is shown with its method (POST, GET, DELETE), URL path, and a corresponding action name (addTicket, createTicketAttribute, deleteTicketAttribute, deleteTicket). Below the ticket section, there is a note about session timeouts and a link to the User Management API. At the bottom, there is a footer with a copyright notice and a feedback button.

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

## Role Based Access Control

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

[Terms of service](#)

[Cisco DevNet](#)

**aaa : APIs to register and manage AAA Servers**

[Show/Hide](#) | [List Operations](#) | [Expand Operations](#) | [Raw](#)

**role : Role Description API**

[Show/Hide](#) | [List Operations](#) | [Expand Operations](#) | [Raw](#)

**ticket : Ticket Management API**

[Show/Hide](#) | [List Operations](#) | [Expand Operations](#) | [Raw](#)

<b>POST</b>	/ticket	addTicket
<b>POST</b>	/ticket/attribute	createTicketAttribute
<b>GET</b>	/ticket/attribute/{attribute}	getIdleTimeout
<b>GET</b>	/ticket/attribute/sessiontimeout	getSessionTimeout
<b>DELETE</b>	/ticket/attribute/{attribute}	deleteTicketAttribute
<b>DELETE</b>	/ticket/{ticket}	deleteTicket

**user : User Management API**

[Show/Hide](#) | [List Operations](#) | [Expand Operations](#) | [Raw](#)

[ BASE URL: <https://sandboxapicem.cisco.com/api/v1/api-docs/rbac-service> , API VERSION: 1.0 ]

[I wish this page would...](#)

# POST /ticket

## Swagger Try it out!

1. Click Model Schema
2. Click the yellow box under Model Schema
3. Enter the DevNet Sandbox APIC-EM credentials between the quotes.
4. Click the “Try it out !” button.
5. If successful, the ticket number will be in the response body JSON.

Module / Swagger API

Parameter	Value	Description	Parameter Type	Data Type
user	{ "password": "Cisco123!", "username": "devnetuser" }	user	body	Model Schema

Parameter content type: application/json

Click to set as parameter value

Error Status Codes

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

Try it out! Hide Response

Request URL

https://sandboxapicem.cisco.com/api/v1/ticket

Response Body

```
{  
    "response": {  
        "serviceTicket": "ST-9749-ACgWKTBxd37b0jzLQ4wv-cas",  
        "idleTimeout": 1800,  
        "sessionTimeout": 21600  
    },  
    "version": "1.0"  
}
```

service ticket

response body JSON

Response Code

200



# Putting It All Together

# Lab - Getting a Service Ticket with Python

# Experimenting with REST APIs using tools

	Swagger API Docs	Postman	Python
Availability	Vendor dependent	Yes	Yes
Ease of use	Easy	Medium	More difficult
Save API calls	No	Yes	Yes
Reuse	No	Yes	Yes
Automation	No	No	Yes
Customization	No	No	Yes
App Integration	No	No	Yes

# Postman

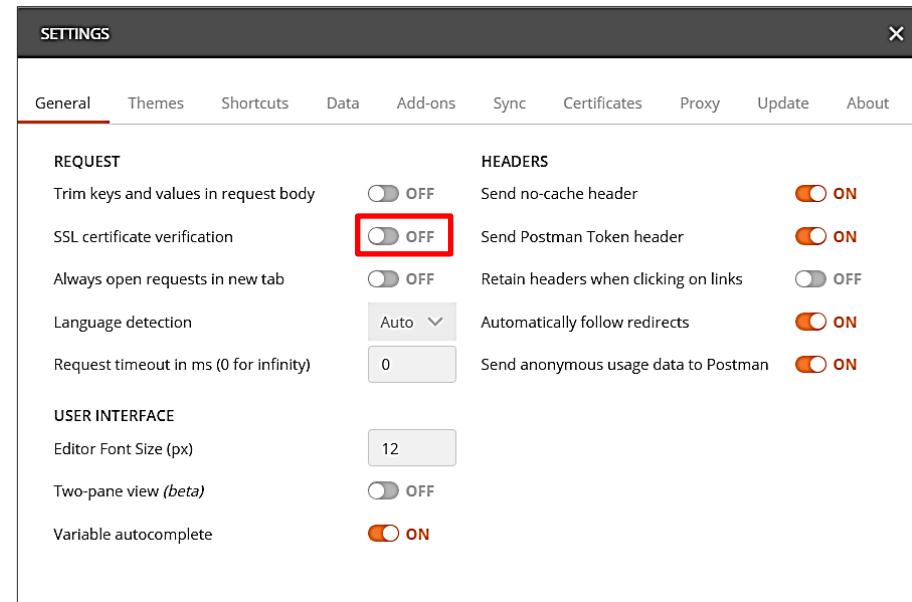
- An app for Windows, Mac OS, or Linux that provides an easy way to interact with REST APIs.
- Allows for headers to be easily constructed.
- Displays request status code and response data.
- Frequently used requests can be saved in tabs, history, or collections for reuse.

# Lab: Getting a Service Ticket with Python

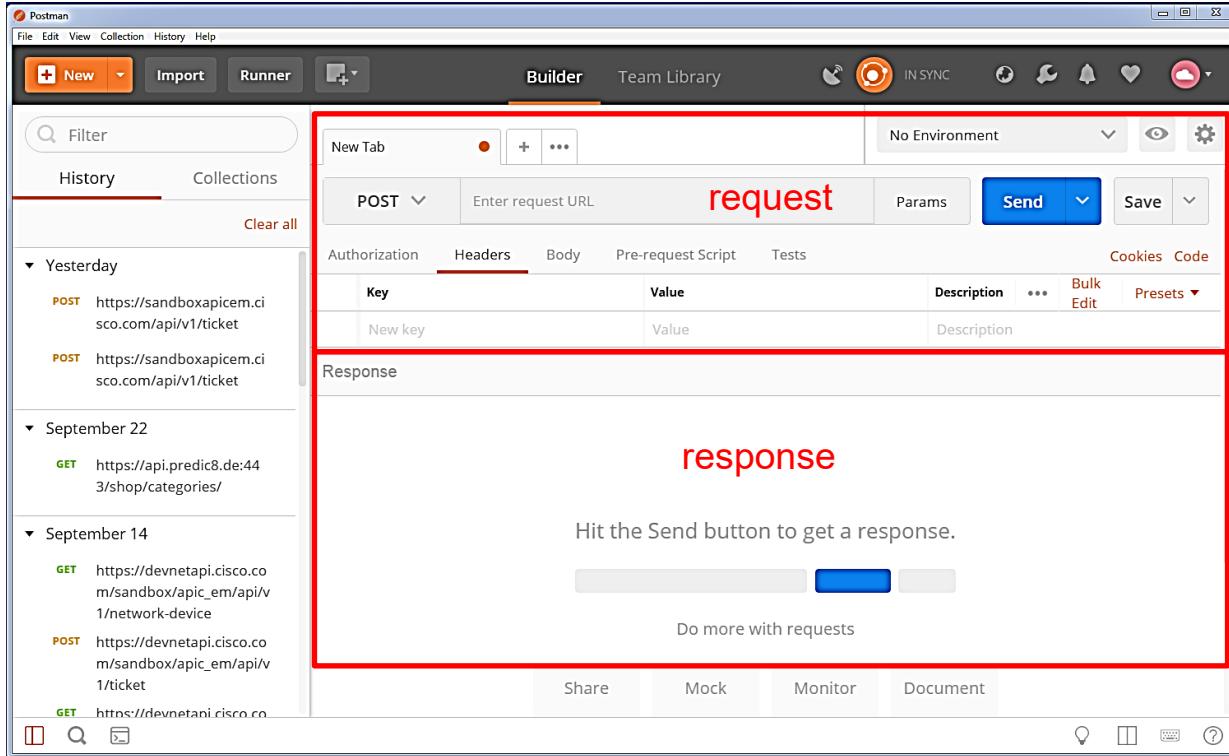
## Part 1 Step 1: Configure Postman

We need to disable SSL certificate checking. This can cause requests to fail.

- Open **File>Settings**.
- In the General tab, under Request, set SSL Certificate Verification to "OFF"



# Postman Features



- History
- Tabs
- Collections
- Presets
- Code
- Environments
- Collaboration

# Lab: Getting a Service Ticket with Python

## Part 1 Steps 2 - 5 Enter Required Information and Send Request

method      URI      Headers

POST  https://sandboxapicem.cisco.com/api/v1/ticket Params

Authorization  Body  Pre-request Script Tests Cookies Code

Key	Value	Description	...	Bulk Edit	Presets ▾
<input checked="" type="checkbox"/> Content-Type	application/json				

Body JSON: {"username": "devnetuser", "password": "Cisco123!"} Body

POST  https://sandboxapicem.cisco.com/api/v1/ticket Params

Authorization   Body  Pre-request Script Tests Cookies Code

form-data  x-www-form-urlencoded  raw  binary JSON (application/json)

1	{"username": "devnetuser", "password": "Cisco123!"}
2	



# Lab: Getting a Service Ticket with Python

## Part 1 Step 5b: View the Response

The screenshot shows a Postman request to `https://sandboxapicem.cisco.com/api/v1/ticket`. The Headers tab is selected, showing a Content-Type header of `application/json`. The Body tab shows a JSON response with an authentication token. The status bar indicates a `200 OK` response.

request status

Status: 200 OK

response body

authentication token  
(service ticket number)

```
1 {  
2   "response": {  
3     "serviceTicket": "ST-9851-QHfCeeVDPxmNToGRcokq-cas",  
4     "idleTimeout": 1800,  
5     "sessionTimeout": 21600  
6   },  
7   "version": "1.0"  
8 }
```

# Overview of the Request Process

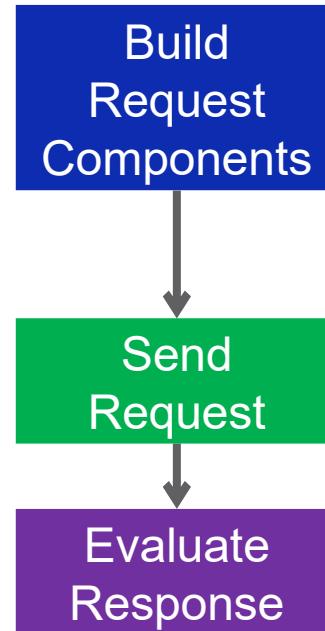
## 1. Build request

- Method
- URL
- Headers
- Body
- Authentication

## 2. Send request

## 3. Evaluate response

- Response code
- Desired data features



# The Path Trace Application Demo

1. Open and run the **04\_path\_trace\_sol.py** file.
2. From the list of devices, enter source and destination IP addresses when prompted and view the resulting output.
3. The application does the following:
  - a) Obtains a service ticket from the APIC-EM **/ticket** endpoint.
  - b) Obtains and displays an inventory of hosts from the **/hosts** endpoint
  - c) Obtains and displays an inventory of network devices from the **/network-devices** endpoint
  - d) Requests source and destination IP addresses for the Path Trace from the user.
  - e) Requests the Path Trace from the **/flow-analysis** endpoint.
  - f) Monitors the status of the Path Trace until it is complete.
  - g) Displays some of the results of the completed Path Trace.
4. We are going to build this!

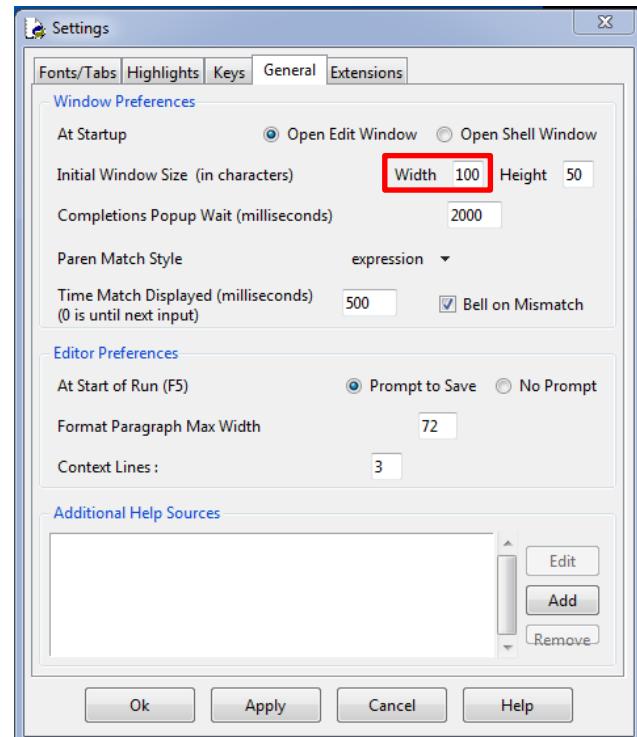
# About Coding...

- Computers are not very smart. You have to tell them **exactly** what to do in a way they understand.
- Common Python “gotchas”:
  - **Punctuation and symbols:** " ", ', :, =, ==, (), {}, [ ] are meaningful and must be used properly.
  - **Paired symbols:** must match (open and close).
  - **Indentation:** standard indentation is four (or multiples of four) spaces. Control structures and functions require indentation.

# Lab 1: Getting a Service Ticket with Python: Part 2 Step 1a-d. Setup the Python Script Environment - 1

The IDLE shell window is a little narrow. In order to get better displays, configure IDLE as follows:

1. Go to the **Options** menu and choose **Configure IDLE**.
2. Go to the **General** tab.
3. Change Initial Window Size to Width = **100**.
4. Click **Ok**.



# Lab 1: Getting a Service Ticket with Python:

## Part 2 Step 1e. Setup the Script Environment

```
"""
01_get_ticket.py
This script retrieves an authentication token from APIC-EM and prints out
it's value. It is standalone, there is no dependency.

MBenson
11/12/2017
"""

import json
import requests

requests.packages.urllib3.disable_warnings()
```

1. Document code with initial comment block.
2. Import required modules: **json** and **requests**
3. Disable SSL certificate warnings

# Lab 1: Getting a Service Ticket with Python: Part 2 Step 2a-c. Build the Request Components

```
api_url = "https://sandboxapicem.cisco.com/api/v1/ticket"  
  
headers = {"content-type": "application/json"}  
  
body_json = {  
    "username": "!!!!REPLACE ME with the username!!!!",  
    "password": "!!!!REPLACE ME with the password!!!!",  
}
```

1. Create a string variable for URL.
2. Create header.
3. Provide body requirements.

**Note:** This is *exactly* what we provided to Postman for the request.

# Lab 1: Getting a Service Ticket with Python:

## Part 2 Step 3. Send the Request

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

1. Create a Python object to hold the response to the request.
2. Provide the variables for the request to the **POST** method of the **requests** module.
3. **json.dumps()** encodes a Python object as JSON.  
This line of code sends the request using a POST method to the URL of APIC-EM ticket endpoint. The response that is returned by the API is stored in the **resp** variable.

**verify=False** is used to turn off SSL verification (OK for learning, NOT OK in production!)

# About the JSON

<https://codebeautify.org/JSONviewer>

The screenshot shows a JSON viewer interface. On the left, the "JSON Input" pane displays the following JSON code:

```
1 {  
2   "response": {  
3     "serviceTicket": "ST-2591-  
4       -UayPt2fHgDIEz5bsBcHD-cas",  
5     "idleTimeout": 1800,  
6     "sessionTimeout": 21600  
7   },  
8   "version": "1.0"  
9 }
```

In the center, the "Result mode:" section includes a dropdown set to "tree", a "Load Url" button, a "Browse" button, a "Tree Viewer" button, a "Default Tab Spac" dropdown, and a "Beautify" button.

The right pane, titled "Result : Tree Viewer", shows the JSON structure as a tree. The "serviceTicket" field under the "response" object is highlighted with a red border. The JSON output is as follows:

```
object {1}  
  array {2}  
    response {3}  
      serviceTicket: ST-2591-  
        -UayPt2fHgDIEz5bsBcHD  
        cas  
      idleTimeout : 1800  
      sessionTimeout : 21600  
    version : 1.0
```

```
response_json = resp.json()  
serviceTicket = response_json["response"]  
["serviceTicket"]
```

# Lab 1: Getting a Service Ticket with Python: Part 2 Step 4. Evaluate the Response

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

response_json = resp.json()

serviceTicket = response_json["response"]["serviceTicket"]

print("The service ticket number is: ", serviceTicket)
```

1. Display response code of request.
2. Decode the JSON **resp** variable into a python object and store in **response\_json** object.
3. Extract the service ticket value from the object.
4. Display service ticket value.
5. Save your file as **get\_ticket.py** and run the code.

# Lab 1: Getting a Service Ticket with Python:

## Step 5. Create a Function from the Program

You will convert your program into a function that can be reused in the future. It will go into a file of APIC-EM utility functions called **my\_apic\_em\_functions.py** that we will create.

Requirements for the function:

1. Defined with **def get\_ticket()** :
2. All subsequent lines of code indented an additional four spaces.
3. This function should return the service ticket number for use in other programs by using a return statement at the end of the function:

```
return serviceTicket
```

# What's next?

- We will create a small program that requests and displays a table of hosts on the network. We convert this to a function and add it to our functions file.
- We will reuse code to create a small program that requests and displays a table of network devices on the network. We convert this to a function and add it to our functions file.
- We will complete code in the Path Trace application and use our functions in that program.

# Lab - Create a Host Inventory in Python

# Lab 2: Create a Host Inventory in Python

## Part 1 Step 1: Setup the Postman Request

The screenshot shows the Postman application interface. At the top, there are two tabs: 'https://sandboxapicem.cisco.com' and 'https://sandboxapicer'. A red box highlights the '+' button in the tab bar, with a callout 'add new tab'. Below the tabs, the method is set to 'GET' and the URL is 'https://sandboxapicem.cisco.com/api/v1/host'. A red box highlights the URL with a callout '/host API endpoint'. To the right are 'Send' and 'Save' buttons. The main area shows the 'Headers (2)' tab selected, containing two entries: 'Content-Type: application/json' and 'X-Auth-Token: ST-2650-aKplXXs6vjNTFdZA7gVvk-cas'. A red box highlights the 'X-Auth-Token' value with a callout 'add your service ticket number'. Other tabs like 'Authorization', 'Body', 'Pre-request Script', 'Tests', 'Cookies', and 'Code' are visible at the bottom.

1. Create a new tab select the method, URI, and Content-Type.
2. Run the Postman tab that obtains a service ticket.
3. Build the REST header with **Content-Type** and service ticket number as the value for **X-Auth-Token**
4. Send request, view response.

# Lab 2: Create a Host Inventory in Python

## Part 1 Step 2: Evaluate the Response

Result : Tree Viewer

```
object {1}
  array {2}
    response [3]
      0 {14}
        hostIP : 10.1.15.117
        hostMac : 00:24:d7:43:59:d8
        hostType : wireless
        connectedNetworkDeviceId : cd6d9b24-839b-4d58-adfe-3fdf781e1782
        connectedNetworkDeviceIpAddress : 10.1.14.3
        connectedAPMacAddress : 68:bc:0c:63:4a:b0
        connectedAPName : AP7081.059f.19ca
        vlanId : 600
        lastUpdated : 1479514114932
        source : 200
        pointOfPresence : ae19cd21-1b26-4f58-8ccd-d265deabb6c3
        pointOfAttachment : ae19cd21-1b26-4f58-8ccd-d265deabb6c3
        subType : UNKNOWN
        id : 48cdeb9b-b412-491e-a80c-7ec5bbe98167
```

## Explore the Response JSON

We want to display a small table of hosts, including the **hostIP** and **hostType** values for each host.

```
response[0]['hostIP']
```

```
response[0]['hostType']
```

# Lab: Create Host Inventory in Python

## Part 2 Step 1: Setup the code environment

```
"""
02_print_hosts.py
gets an inventory of hosts from /host endpoint
November, 2017
"""

import requests
import json
from tabulate import *
from my_apic_em_functions import *
```

1. Document
2. Import required modules

Note that the Python file that contains your service ticket function is imported for use here. The name of the functions file will vary depending on whether you are using your own file or the provided solution file.  
Save your new code file as **print\_hosts.py**

# Lab: Create Host Inventory in Python

## Part 2 Step 2: Build Request Components

```
api_url = "https://{{YOUR-APICEM}}.cisco.com/api/v1/host"

ticket = get_ticket()
headers = {
    "content-type": "application/json",
    "X-Auth-Token": ticket
}
```

Note that the `get_ticket()` function that you created earlier is reused here and the value is supplied to the `headers` object.

# Lab: Create Host Inventory in Python

## Part 2 Step 3: Make the Request and Handle Errors

```
resp = requests.get(api_url, headers=headers, verify=False)
print ("Status of /host request: ", resp.status_code)
if resp.status_code != 200:
    raise Exception("Status code is not 200/OK. Response text: " + resp.text)
response_json = resp.json() # Get the json-encoded content from response
```

1. Request is made with the **get()** method of the **requests** module.
2. A message is displayed with the status code of the request.
3. If the status code is not 200 (OK), stop the execution by raising an Exception with the explanatory message and the response text.

# Lab: Create Host Inventory in Python

## Part 2 Step 4: Evaluate the Response

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

table_header = ["Number", "Type", "IP"]
print ( tabulate(host_list, table_header) )
```

1. The **for:** loop iterates through the objects in **response\_json[response]** key, which corresponds to each host.
2. The data for the host is put in the variable **item**.
3. This variable contains all the keys for the host.
4. We extract the "**hostType**", and "**hostIp**" for each host.
5. Each iteration of the loop appends this information to a new line in the variable.
6. We pass the **host\_list** variable to **tabulate** to be formatted and print the result.

# Lab: Create Host Inventory in Python

## Part 2 Step 5: Create the Function

```
def print_hosts():
    api_url = "https://sandboxapicem.cisco.com/api/v1/host"

    ticket = get_ticket()
    headers = {"content-type": "application/json", "X-Auth-Token": ticket}

    resp = requests.get(api_url, headers=headers, verify=False)
    print ("Status of /host request: ", resp.status_code)
    if resp.status_code != 200:
        raise Exception("Status code is not 200/OK. Resp text: " +
resp.text)
    response_json = resp.json()

    host_list=[]
    i=0
    for item in response_json["response"]:
        i+=1
        host = [ i, item["hostType"], item["hostIp"] ]
        host_list.append( host )
    table_header = [ "number", "type", "host IP" ]
    print( tabulate(host_list, table_header) )
```

1. Copy your program into the functions file (`my_apic_em_functions.py`).
2. Define the function with `def print_hosts()`:
3. Indent everything by four *additional* spaces
4. Save the functions file.

# Lab - Create a Network Device Inventory in Python

# Lab: Create a Network Device Inventory in Python

Replicate your work for the **/network-device** Inventory endpoint.

1. Save your `print_hosts.py` file as `print_devices.py`.
2. Go to the APIC-EM GUI and open the Swagger page for the inventory/network-device
3. Click try it and look at the returned JSON.
4. We want to access and print '`type`' and '`managementIpAddress`' instead of "`hostType`" and "`hostIp`".
5. Inspect the code and make the substitutions everywhere they are required. Some variable names may need to change to make sense in the context of this script.
6. Save the file and test. Add the function `print_devices()` to your functions file.

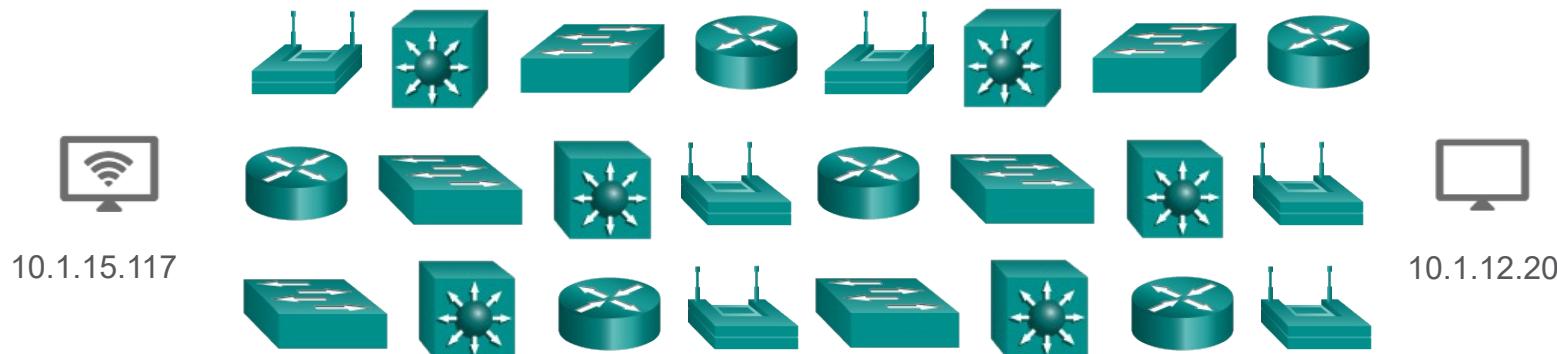
# Lab - Using the APIC-EM Path Trace API

# APIC-EM Path Trace

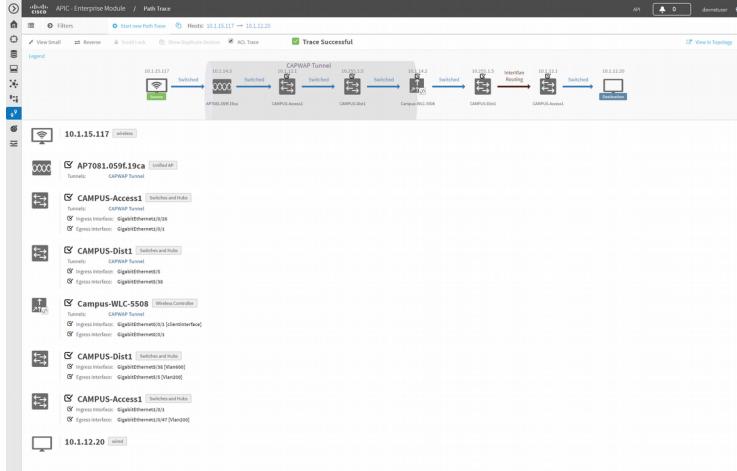


What is the detailed network path\*  
between 10.1.15.117 and 10.1.12.20?

\*detailed network path = APs, Switches, Routers + ACLs, QoS, Statistics, etc.



# GUI: Path Trace between 10.1.15.117 and 10.1.12.20



## Use case:

How to optimize the Network Helpdesk team operations with the APIC-EM Path Trace?

*E.g. user reports being unable to access from his PC the File server.*



Could we optimize even more the operations of the Network Helpdesk team with API integrations between the Helpdesk app and APIC-EM using a simple script?

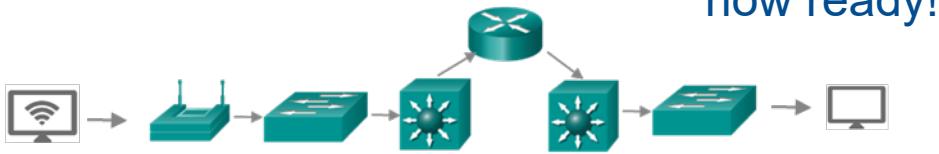
# API: Path Trace between 10.1.15.117 and 10.1.12.20

1. Send a request  
using the APIC-EM  
Path Trace API

2. Working on it!  
Here is the ID of the job.

3. Is the ID job done?

4. {JSON} ← The output is  
now ready!



10.1.15.117

10.1.12.20



# Lab: Using the APIC-EM Path Trace API: Process

- You will work from partially completed code in the **04\_path\_trace.py** file.
- Copy and paste from what you have already completed.
- Consult the solution files.
- Seek assistance from the workshop community if you are stuck.
- Coders collaborate, so should you!

# Lab: Using the APIC-EM Path Trace API: About the working code file

- Open the **04\_path\_trace.py** work file in IDLE.
- The code is divided into six sections. The lab references each section.
- You are directed to complete or supply statements in the code.
- Some material is new. The lab document provides information regarding what is required.
- You are working on a functioning application. Sometimes it is necessary to use code that is more advanced than your current skill level. You are not expected to understand that code, although it can be explained at a later time if you wish.

# Lab: Using the APIC-EM Path Trace API: Testing your code...

- In IDLE, create a new Python file called **test.py**.
- Save it in the same folder as your other lab files.
- As you complete a section of code, copy and paste it into this file, save, and run it.

# Lab: Using the APIC-EM Path Trace API:

## Section 1: Setup the Environment

```
#=====
# Section 1. Setup the environment and variables required
# to interact with the APIC-EM
#=====
#+++++++Add Values+++++
#import modules
[REDACTED]
#disable SSL certificate warnings
[REDACTED]
#++++++++
#+++++++Add Values+++++
# Path Trace API URL for flow_analysis endpoint
api_url = [REDACTED] #URL of API endpoint
# Get service ticket number using imported function
ticket = [REDACTED] # Add function to get service ticket
# Create headers for requests to the API
headers = [REDACTED] # Create dictionary containing headers for
the request
#++++++++
[REDACTED]
```

Add code where indicated to setup the code environment and build the request components.

# Lab: Using the APIC-EM Path Trace API

## Code Section 2: Display list of hosts and devices

```
#=====
# Section 2. Display list of devices and IPs by calling
print_hosts() and print_devices()
=====

#+++++Add Values+++++
#print message identifying that the list of hosts
follows
[REDACTED]

# Call your function name that displays hosts
[REDACTED]

#print message identifying that the list of devices
follows
[REDACTED]

# Add function to display network devices
[REDACTED]

#+++++
```

Print short text messages that identify to the user what is to be displayed.

Use your **print\_hosts()** and **print\_devices()** functions here.

# Lab: Using the APIC-EM Path Trace API

## Code Section 3: Get Source and Destination IP Addresses from User

```
while True:  
    #####Add Values#####  
    s_ip = [REDACTED] # Request user input for source IP address  
    d_ip = [REDACTED] # Request user input for destination IP address  
    #####  
    #Various error traps could be completed here - POSSIBLE CHALLENGE  
  
    if s_ip != "" or d_ip != "":  
        path_data = {  
            "sourceIP": s_ip,  
            "destIP": d_ip  
        }  
        break #Exit loop if values supplied  
    else:  
        print("\n\nYOU MUST ENTER IP ADDRESSES TO CONTINUE.\nUSE CTRL-C TO QUIT\n")  
        continue #Return to beginning of loop and repeat
```

*variable = input("prompt: ")*

# Lab: Using the APIC-EM Path Trace API

## Code Section 4: Initiate the Path Trace and get the Flow Analysis ID

```
#=====
# Section 4. Initiate the Path Trace and get the flowAnalysisId
#=====

#+++++++Add Values+++++++
#Convert the path data dictionary to JSON using json.dumps(). Store in path
path = [REDACTED]
# Post request to initiate Path Trace
# Make the request. Construct the POST request with the json formatted path data to the API
resp = [REDACTED]
# Inspect the return, get the Flow Analysis ID, put it into a variable
resp_json = resp.json()
Assign the value of the "flowAnalysisID" key of the "response" array of resp_json
flowAnalysisId = flowAnalysisId = resp_json["response"] [REDACTED]
#++++++++
print("FLOW ANALYSIS ID: ", flowAnalysisId)
```

# Lab: Using the APIC-EM Path Trace API

## Code Section 5: Check status of Path Trace request - 1

```
#=====
# Section 5. Check status of Path Trace request, output results when COMPLETED
#=====

#++++++Add Values+++++
# Add Flow Analysis ID to URL in order to check the status of this specific path trace
# Append "/" + flowAnalysisId to the flow analysis end point API URL that was
# created in Section 1
check_url = [REDACTED]

#+++++
```

# Lab: Using the APIC-EM Path Trace API

## Code Section 5: Check status of Path Trace request - 2

```
status = ""
checks = 0 #variable to increment within the while loop. Will trigger exit from loop after x iterations
while status != "COMPLETED":
    checks += 1
    r = requests.get(check_url, headers=headers, verify=False)
    response_json = r.json()
    #####Add Values#####
    status = response_json["response"]["request"][[ ] ] # Assign the value of the status
    of the path trace request from response_json
    #####
    #wait one second before trying again
    time.sleep(1)
    if checks == 15: #number of iterations before exit of loop; change depending on conditions
        raise Exception("Number of status checks exceeds limit. Possible problem with Path Trace.")
        #break
    elif status == "FAILED":
        raise Exception("Problem with Path Trace")
        #break
    print("REQUEST STATUS: ", status) #Print the status as the loop runs
```

# Lab: Using the APIC-EM Path Trace API

## Code Section 5: Check status of Path Trace request - 3

### JSON Status Key

```
status = response_json["response"]["request"]["status"]
```

JSON Input

```
1 [{}  
2   "response": {  
3     "request": {  
4       "sourceIP": "10.1.15.117",  
5       "destIP": "10.2.1.22",  
6       "periodicRefresh": false,  
7       "id": "9c75b61d-2795-43b4-a6a9-d756c17a1da8",  
8       "status": "COMPLETED",  
9       "createTime": 1510154827375,  
10      "lastUpdateTime": 1510154834878  
11    },  
12    "lastUpdate": "Wed Nov 08 15:27:55 UTC 2017",  
13    "networkElementsInfo": [  
14      {  
15        "id": "48cded9b9b-b412-491e-a80c-7ec5bbe98167",  
16        "type": "wireless",  
17        "ip": "10.1.15.117",  
18        "linkInformationSource": "Switched"  
19      },  
20      {  
21        "id": "cd6d9b24-839b-4d58-adfe-3fdf781e1782",  
22        "name": "AP7081.059f.19ca",  
23        "type": "Unified AP",  
24        "ip": "10.1.14.3",  
25        "role": "ACCESS",  
26        "linkInformationSource": "Switched",  
27        "tunnels": []  
28      }  
29    ]  
30  }]
```

Result : Tree Viewer

```
object {1}  
  array {2}  
    response {4}  
      request {7}  
        sourceIP : 10.1.15.117  
        destIP : 10.2.1.22  
        periodicRefresh : false  
        id : 9c75b61d-2795-43b4-a6a9-d756c17a1da8  
        status : COMPLETED  
        createTime : 1510154827375  
        lastUpdateTime : 1510154834878  
        lastUpdate : Wed Nov 08 15:27:55 UTC 2017  
      networkElementsInfo [12]  
      detailedStatus {1}  
      version : 1.0
```

# Lab: Using the APIC-EM Path Trace API

## Code Section 6: Display Results

```
#=====
# Section 6. Display results
#=====

# Create required variables
#++++++Add Values+++++
path_source = [REDACTED] #Assign the source address from the trace from response_json
path_dest = [REDACTED] #Assign the destination address from the trace from response_json
networkElementsInfo = [REDACTED] #Assign the list of all network element dictionaries from response_json
#+++++
```

Supplying these values requires parsing the Path Trace JSON that is has been converted to Python objects and is stored in `response_json`. We will explore an example of the Path Trace JSON now.



# Lab: Using the APIC-EM Path Trace API

## JSON Practice - View Tree

1. Open the **json\_data.json** file that is in the folder with the lab Python files.
2. Copy the entire contents of the file.
3. Open JSON Viewer and paste the JSON in the left-hand pane.
4. View as a tree.
5. Collapse all levels.

# Lab 4 Using the APIC-EM Path Trace API: JSON Practice - Tree View

## JSON Viewer

Save & Share

JSON Input sample

```
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
        },
        "role": "CORE",
        "linkInformationSource": "ECMP"
    },
    {
        "id": "55450140-de19-47b5-ae80
            -bfd741b23fd9",
        "name": "CAMPUS-Router2",
        "type": "Routers",
        "ip": "10.1.4.2",
        "ingressInterface": {
            "physicalInterface": {
                "id": "20c9326c-82b0-47b9-aee5
                    -f57b7725bd01",
                "name": "GigabitEthernet0/0/1"
            }
        },
        "egressInterface": {
            "physicalInterface": {
                "id": "e595740b-38fd-4c87-9cb3
                    -407672ed9dc5",
                "name": "GigabitEthernet0/0/3"
            }
        },
        "role": "BORDER ROUTER",
        "linkInformationSource": "OSPF"
    },
    {
        "id": "UNKNOWN",
        "name": "UNKNOWN",
        "ip": "UNKNOWN",
        "role": "UNKNOWN",
    }
```

Result mode: tree ▾

Load Url

Browse

Tree Viewer

Default Tab Space ▾

Beautify

Result : Tree Viewer

```
object {1}
  array {2}
    response {4}
      request {7}
        lastUpdate : Fri Sep 29 14:03:43 UTC 2017
      networkElementsInfo [12]
      detailedStatus {1}
    version : 1.0
```

Ad closed by Google

Stop seeing this ad

AdChoices ▾

# Lab: Using the APIC-EM Path Trace API

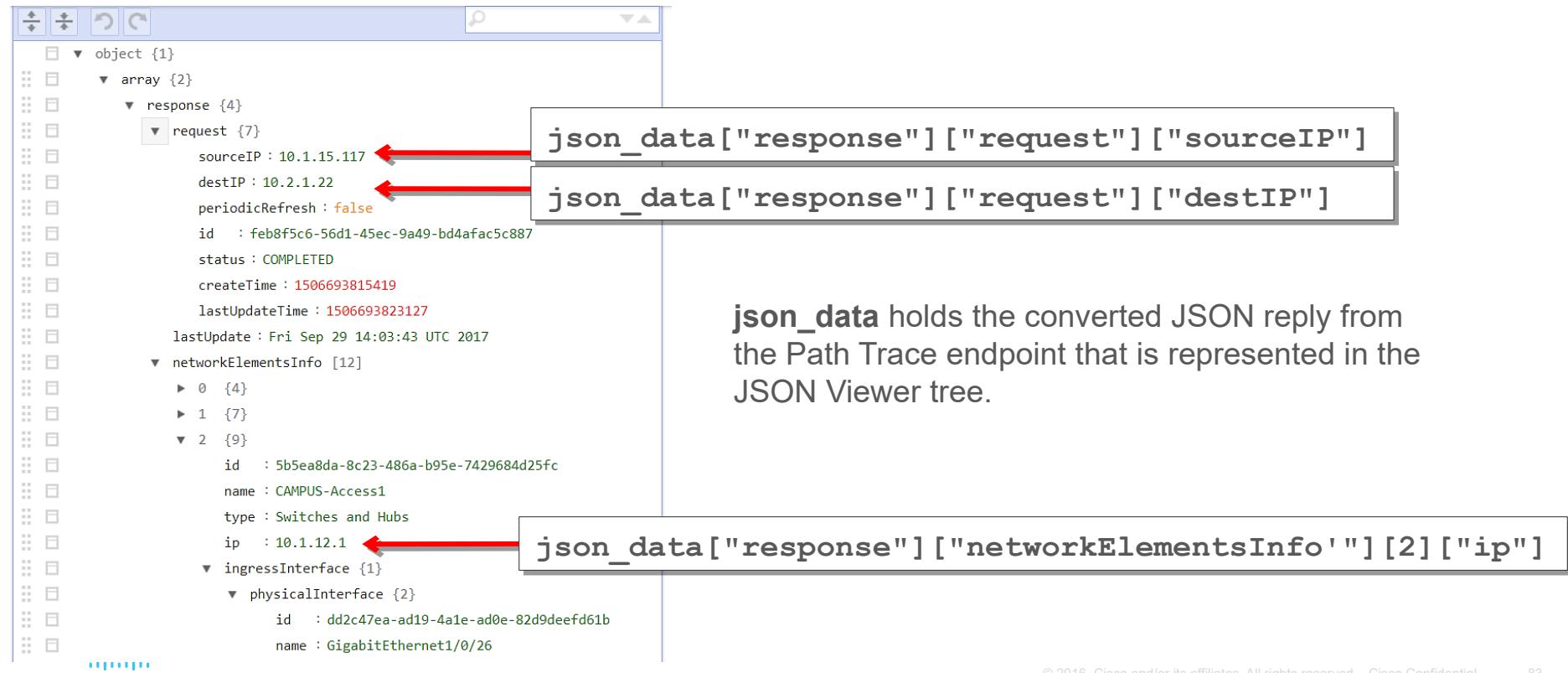
## JSON Practice - Load Variables

```
import json
json_data=json.load(open("path_trace_json.json"))
>>> print(json_data)
```

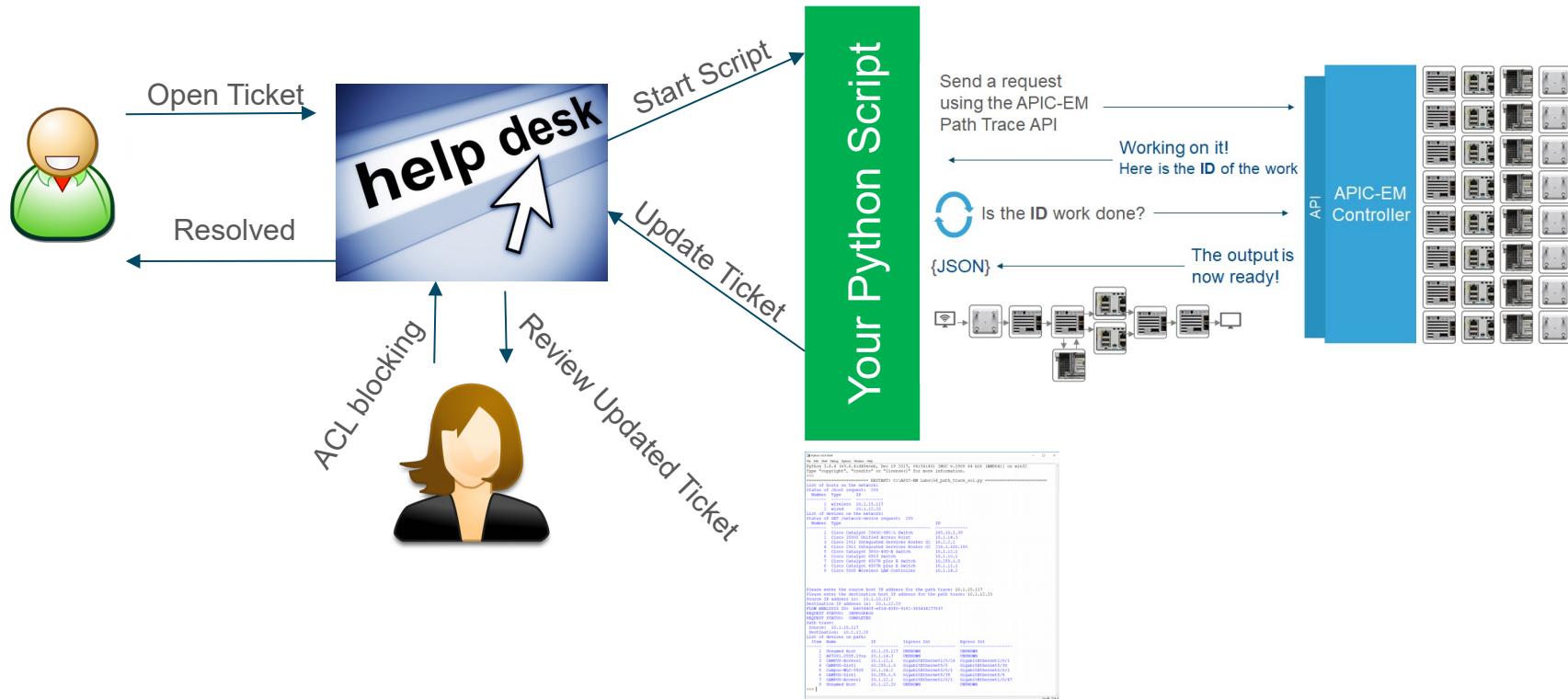
1. Import the **json** module.
2. Open the **path\_trace\_json.json** file, convert it to Python objects, and assign the result to a variable called **json\_data** as shown above.
3. Save and run the program.
4. Display the contents of **json\_data** in the shell. This is what the imported and converted JSON looks like to Python
5. Display the values of different keys in the json. Example:

```
print(json_data['response']['request'])
```

# Lab: JSON Practice - Accessing Data in the Response



# Lab summary



# APIC-EM Tools DevNet GitHub Collection\_

<https://github.com/CiscoDevNet/apic-em-samples-aradford/tree/master/tools/postman>

The screenshot shows a GitHub repository page for 'CiscoDevNet / apic-em-samples-aradford'. The repository has 11 pull requests, 1 star, and 1 fork. The master branch is selected. A red box highlights two commit messages:

- APIC-EM%20Sandbox.json.postman\_collection Updated password and added extra network-device examples.
- APIC-EM%20Sandbox.postman\_environment Updated password and added extra network-device examples.

Below the commits, there are sections for 'Introduction' and 'Importing'.

**Introduction**

To use these examples, you need the latest version of postman (3.2.0). It has free Jetpacks support. This is required for linking requests.

**Importing**

You need to import the collection as well as the environment. You can import the raw git files using the **From URL** option, as seen below. You can also download them and import from file/folder.

Can be imported into Postman as files, or directly from URLs. See the README.md file for more information.

**Note:** Before posting any code your own repository, remove any confidential information from the code and replace it with comments or descriptive placeholder text.

# Next steps...

- Go to DevNet and investigate:
  - The DevNet Cisco Community
  - The DevNet Introduction to DevNet interactive course track
  - The APIC-EM Sandbox and Swagger API documentation

