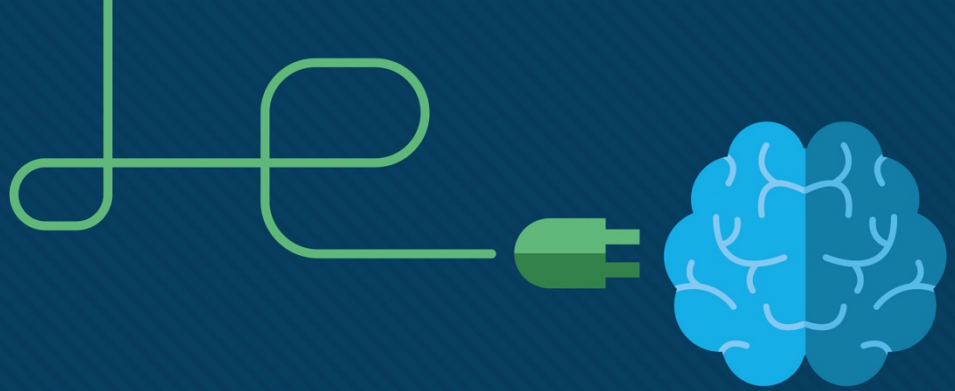


Cisco Networking Academy

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

Name
Title
Date

Network Programmability with Cisco APIC-EM

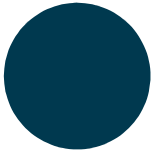
Chapter 2: Programming the APIC-EM

Presenter Name
Presenter Info (Title, School, etc.)
Date

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.

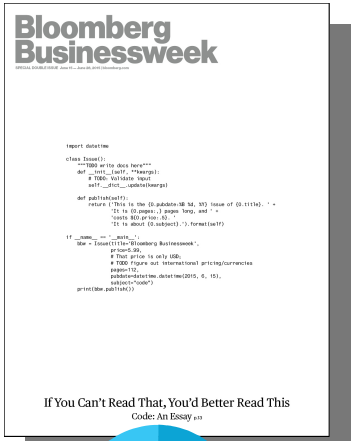
Please note: You are NOT expected become software developers or network programmers - yet!

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**

Building an Industry Ecosystem with DevNet

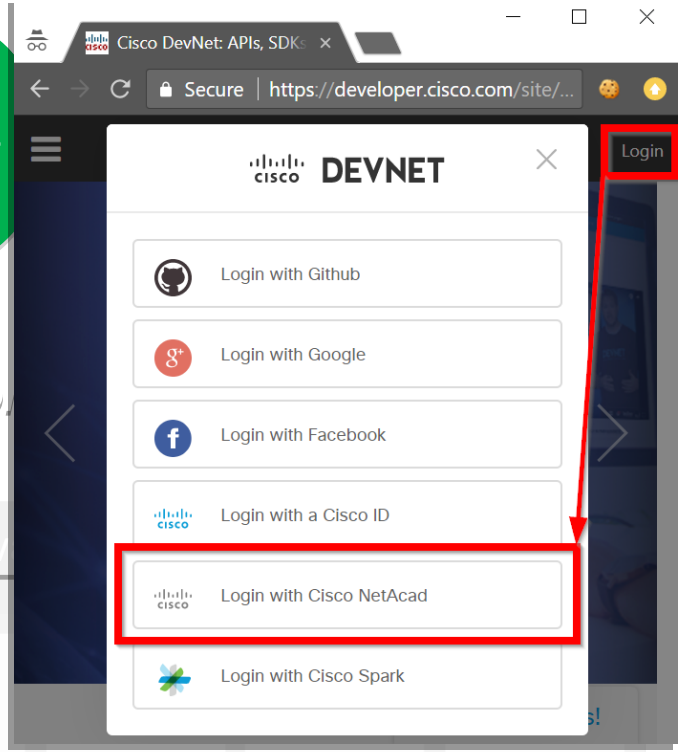
Enabling developers and learners

Developer as the customer

Cisco's Developer Community and Innovation Ecosystem

Cisco's portfolio as a Platform for Innov

https://developer.cisco.com/



IoT | Cloud | Networking | Data Center | Security | Analytics & Automation | Open Source | Collaboration | Mobility

Network Programmability

Network Programmability Concepts

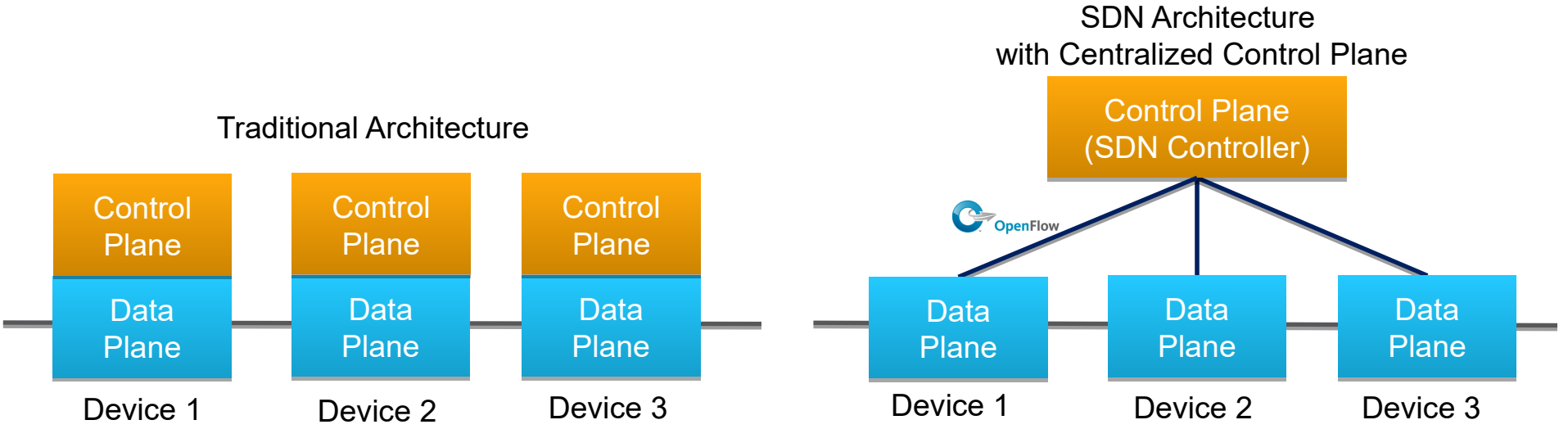
SDN: Control Plane and Data Plane

Control Plane

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

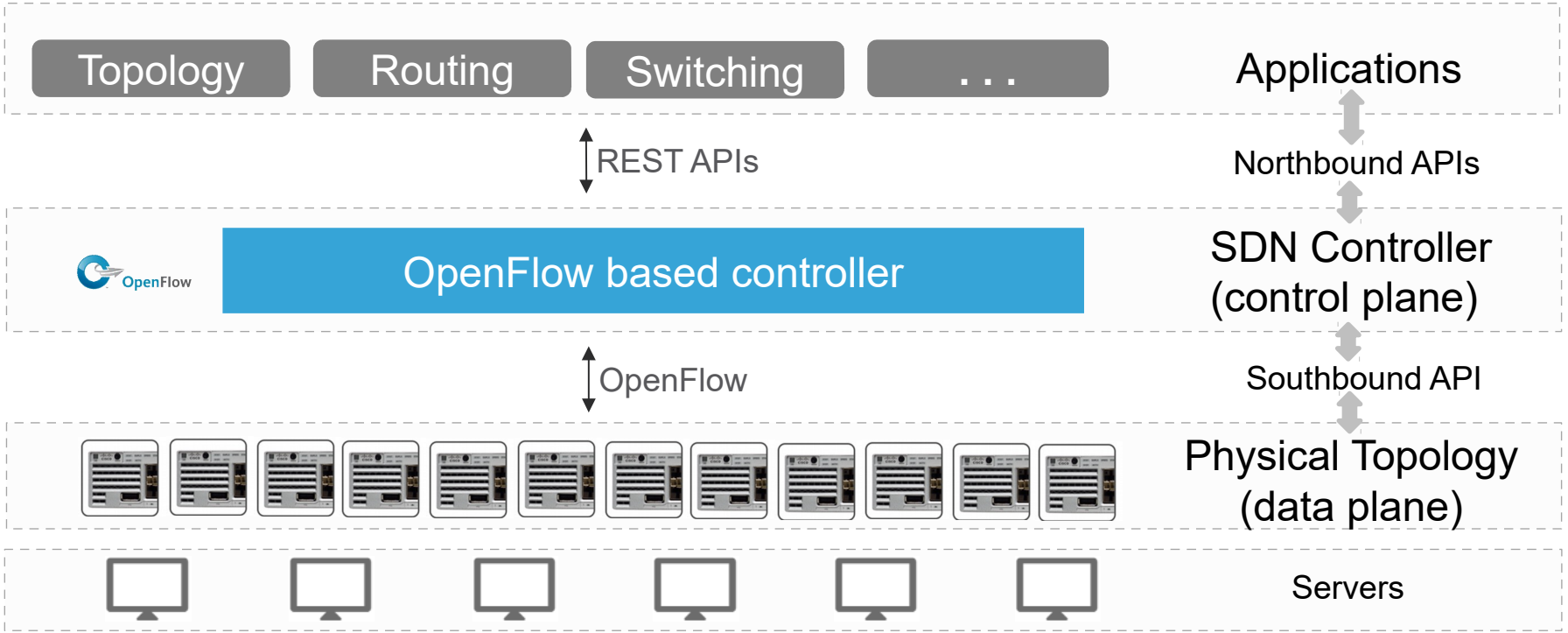
Hardware	Purpose	Example Processes
Dedicated ASICs	forwards traffic to the selected destination	packet switching, L2 switching, QoS, policies, ACLs

Traditional and SDN Architectures

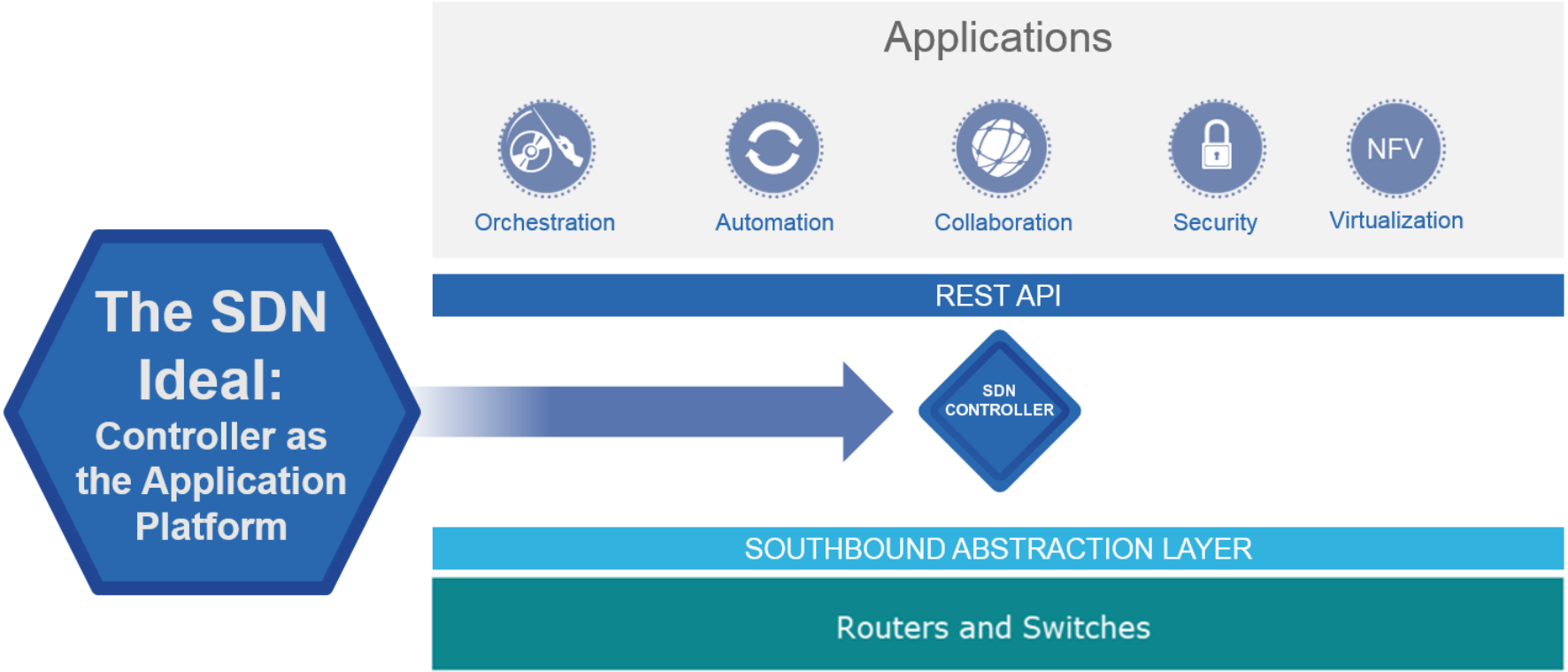


OpenFlow is a protocol between SDN controllers and network devices, as well as a specification of the logical structure of the network switch functions.

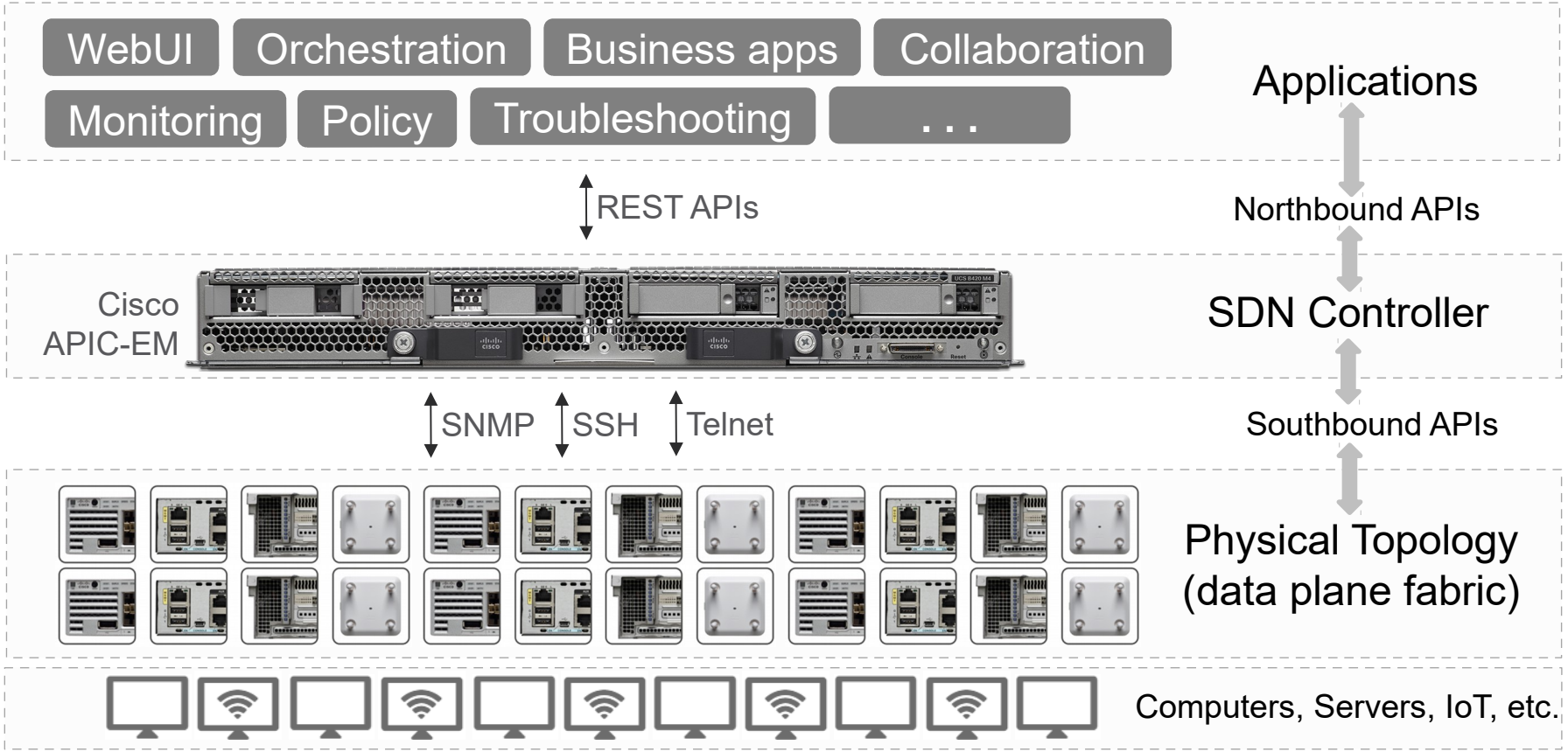
OpenFlow SDN Model



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):

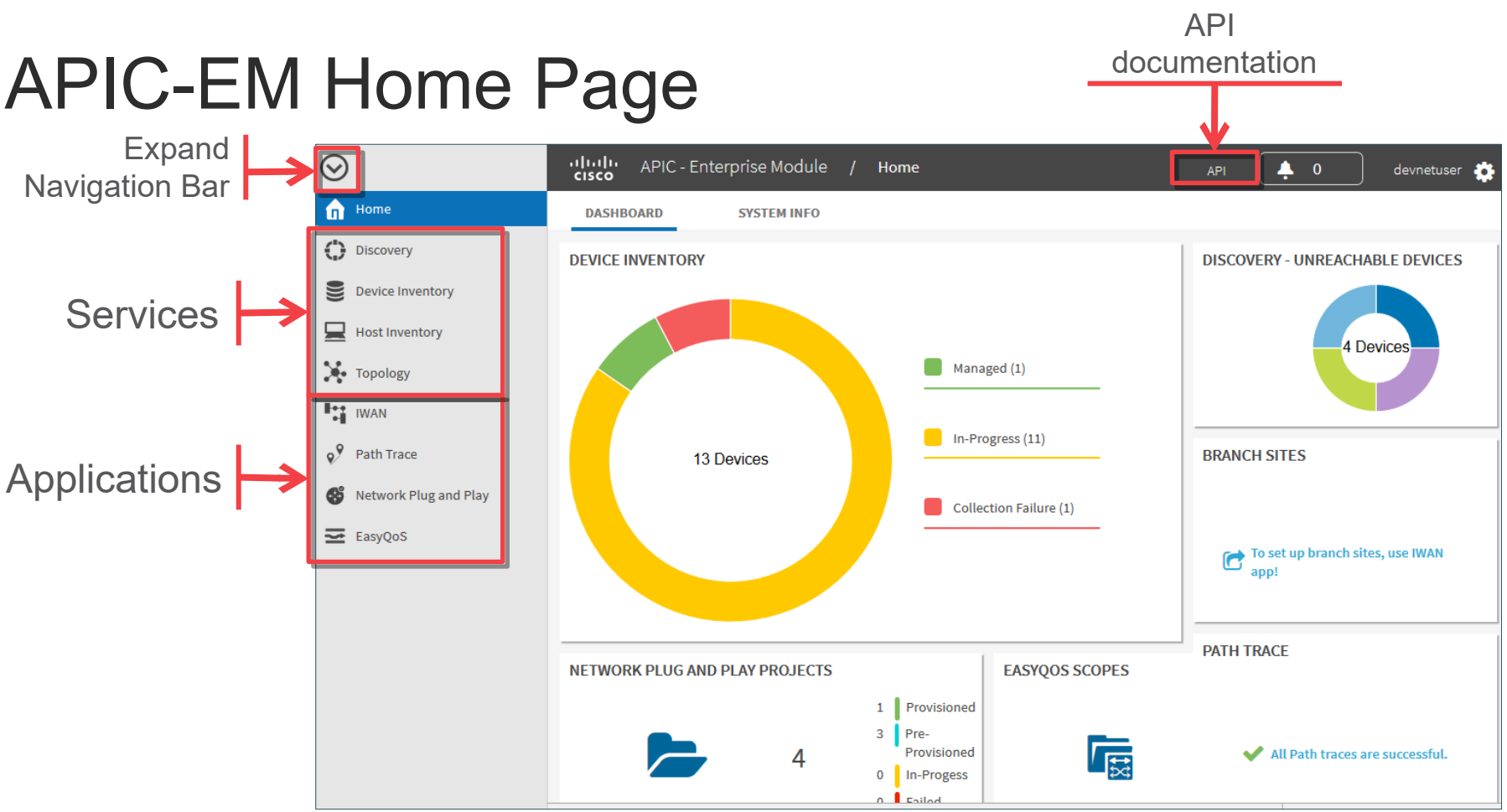
- A Software-Defined Networking (SDN) controller for enterprise networks
- A virtual, software-only, or physical appliance
- Creates an intelligent, open, programmable network with open APIs
- Can transform business-intent policies into dynamic network configuration
- Provides a single point for network-wide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

APIC-EM – Log in

https://sandboxapicem.cisco.com/

User: **devnetuser**
P/W: **Cisco123!**

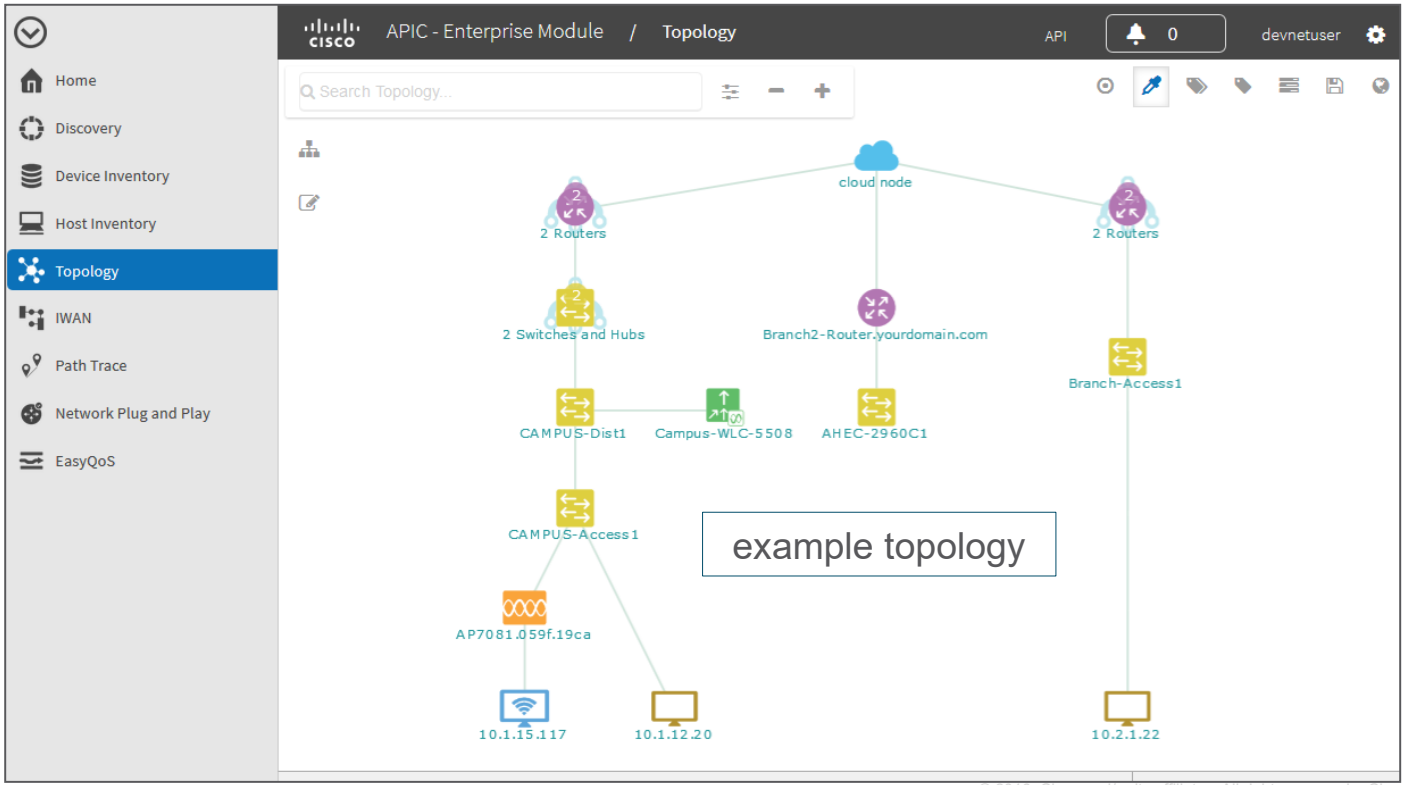
APIC-EM Home Page



APIC-EM Applications

- **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) Application**
  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.

APIC-EM Topology Page



Programming the APIC-EM REST API

REST APIs

What is so great about REST*?

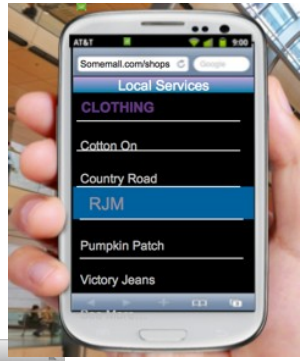
Cisco APIC-EM REST APIs

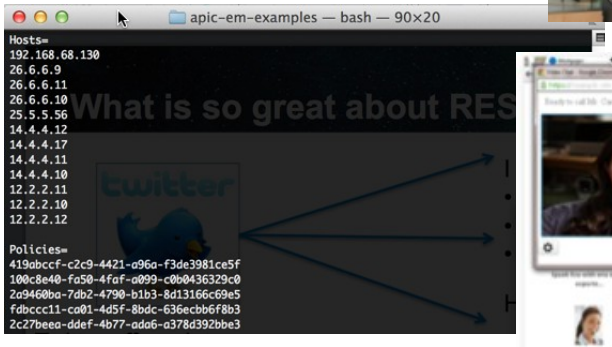
- Hosts
- Devices
- Users
- + more

How does this work?

Easy to use:

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





*representational state transfer (REST).

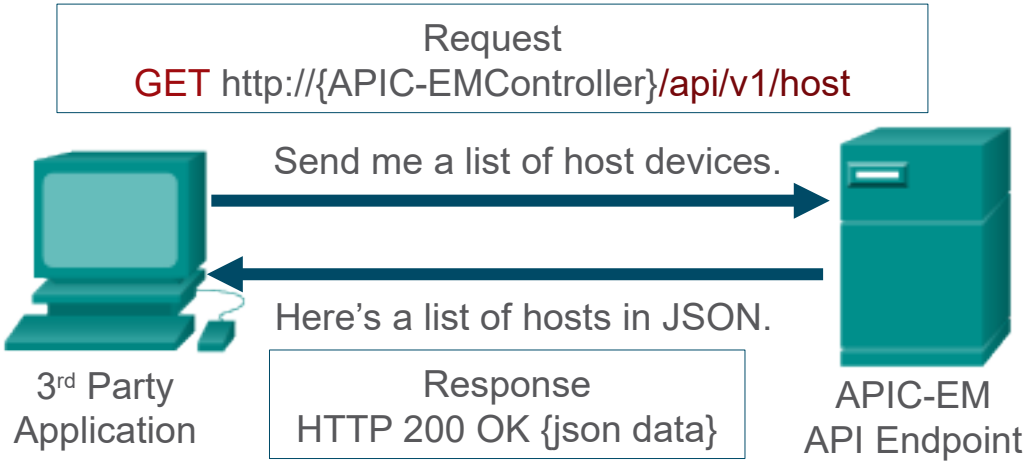
25

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

28

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

Example output of a HTTP response in the Postman application

29

JSON and XML

JSON

sample


```
{
    "response": {
        "request": {
            "sourceIP": "10.1.15.117",
            "destIP": "10.2.1.22",
            "periodicRefresh": false,
            "id": "feb8f5c6-56d1-45ec-9a49-bd4afac5c887",
            "status": "COMPLETED",
            "createTime": 1506693815419,
            "lastUpdateTime": 1506693823127
        },
        "lastUpdate": "Fri Sep 29 14:03:43 UTC 2017",
        "networkElementsInfo": [
            {
                "id": "48cdeb9b-b412-491e-a80c-7ec5bbe98167",
                "type": "wireless",
                "ip": "10.1.15.117",
                "linkInformationSource": "Switched"
            },
            {
                "id": "cd6d9b24-839b-4d58-adfe-3fdf781e1782",
                "name": "AP7081.059f.19ca",
                "type": "Unified AP",
                "ip": "10.1.14.3",
                "role": "ACCESS",
                "linkInformationSource": "Switched",
                "tunnels": [
                    "CAPWAP Tunnel"
                ]
            },
            {
                "id": "5b5ea8da-8c23-486a-b95e-7429684d25fc",
                "name": "CAMPUS-Access1",
                "type": "Switches and Hubs",
                "ip": "10.1.12.1",
                "ingressInterface": {
                    "physicalInterface": {
                        "id": "dd2c47ea-ad19-4a1e-ad0e-82d9deefd61b",
                        "name": "GigabitEthernet1/0/26"
                    }
                },
                "egressInterface": {
                    "physicalInterface": {
                        "id": "38c72319-855e-43bc-8458-94f695d435b6",
                        "name": "GigabitEthernet1/0/1"
```


XML


```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
    <request>
        <sourceIP>10.1.15.117</sourceIP>
        <destIP>10.2.1.22</destIP>
        <periodicRefresh>false</periodicRefresh>
        <id>feb8f5c6-56d1-45ec-9a49-bd4afac5c887</id>
        <status>COMPLETED</status>
        <createTime>1506693815419</createTime>
        <lastUpdateTime>1506693823127</lastUpdateTime>
    </request>
    <lastUpdate>Fri Sep 29 14:03:43 UTC 2017</lastUpdate>
    <networkElementsInfo>
        <id>48cdeb9b-b412-491e-a80c-7ec5bbe98167</id>
        <type>wireless</type>
        <ip>10.1.15.117</ip>
        <linkInformationSource>Switched</linkInformationSource>
    </networkElementsInfo>
    <networkElementsInfo>
        <id>cd6d9b24-839b-4d58-adfe-3fdf781e1782</id>
        <name>AP7081.059f.19ca</name>
        <type>Unified AP</type>
        <ip>10.1.14.3</ip>
        <role>ACCESS</role>
        <linkInformationSource>Switched</linkInformationSource>
        <tunnels>CAPWAP Tunnel</tunnels>
    </networkElementsInfo>
    <networkElementsInfo>
        <id>5b5ea8da-8c23-486a-b95e-7429684d25fc</id>
        <name>CAMPUS-Access1</name>
        <type>Switches and Hubs</type>
        <ip>10.1.12.1</ip>
        <ingressInterface>
            <physicalInterface>
                <id>dd2c47ea-ad19-4a1e-ad0e-82d9deefd61b</id>
                <name>GigabitEthernet1/0/26</name>
            </physicalInterface>
        </ingressInterface>
        <egressInterface>
            <physicalInterface>
                <id>38c72319-855e-43bc-8458-94f695d435b6</id>
                <name>GigabitEthernet1/0/1</name>
            </physicalInterface>
        </egressInterface>
```

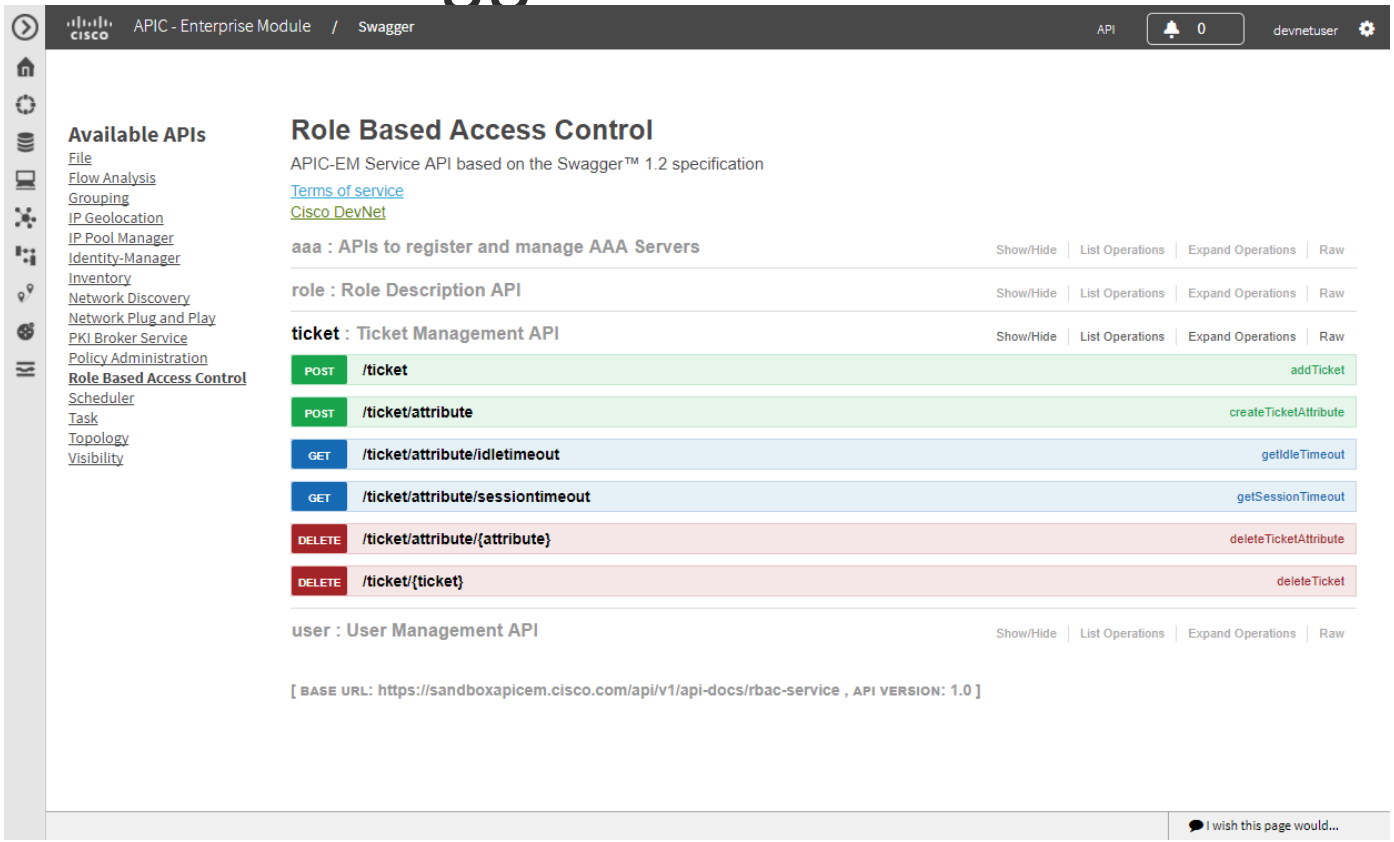
What about authentication?

- **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. The token is used to identify the user and rights when making APIs calls in order to verify access and control.
- **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.

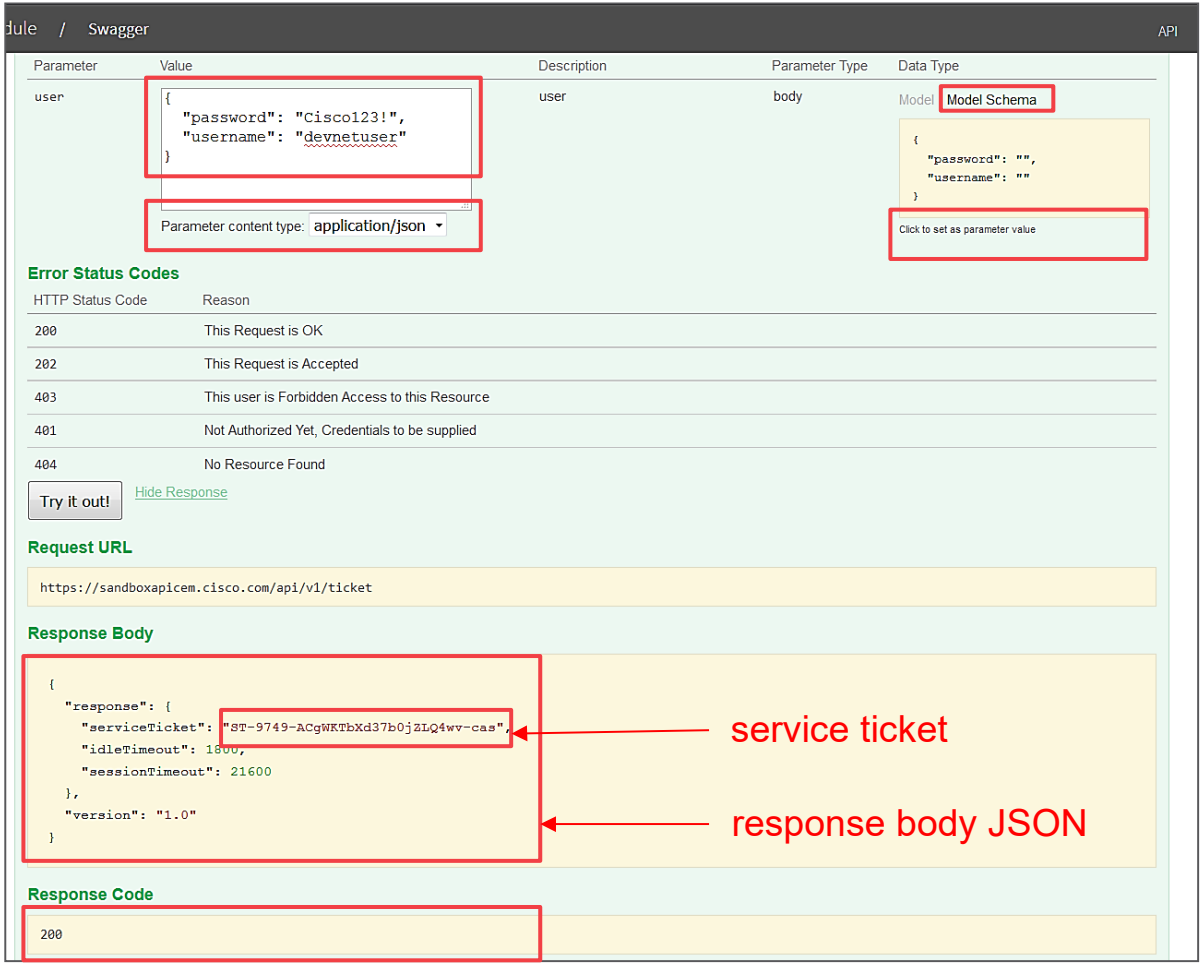
31

APIC-EM Swagger Documentation



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.



Putting It All Together

Lab - Getting a Service Ticket with Python

Experimenting with REST APIs using tools

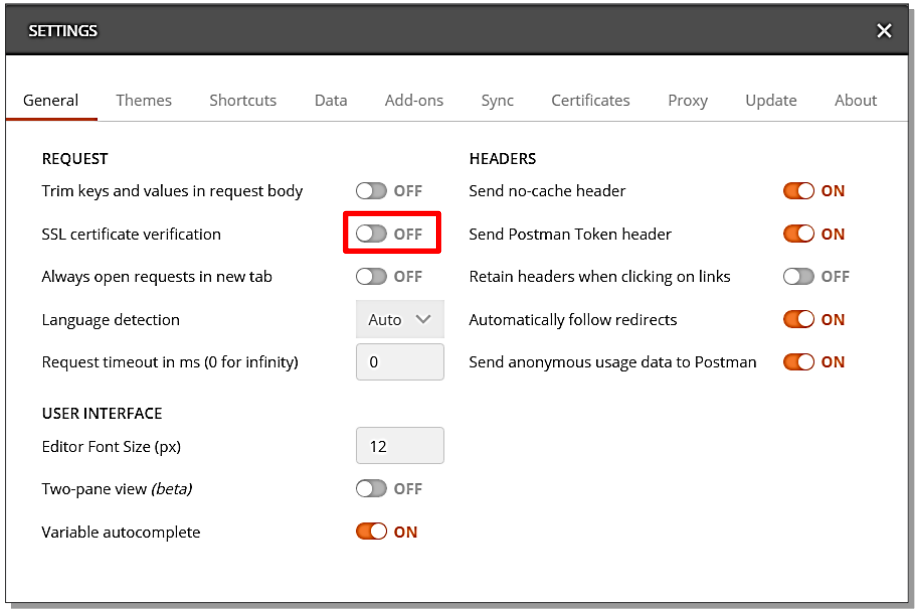
	Swagger API Docs	Postman	Python
Availability	Vendor dependent	Yes	Yes
Ease of use	Easy	Medium	More difficult
Save API calls	No	Yes	Yes
Reuse	No	Yes	Yes
Automation	No	No	Yes
Customization	No	No	Yes
App Integration	No	No	Yes

Postman

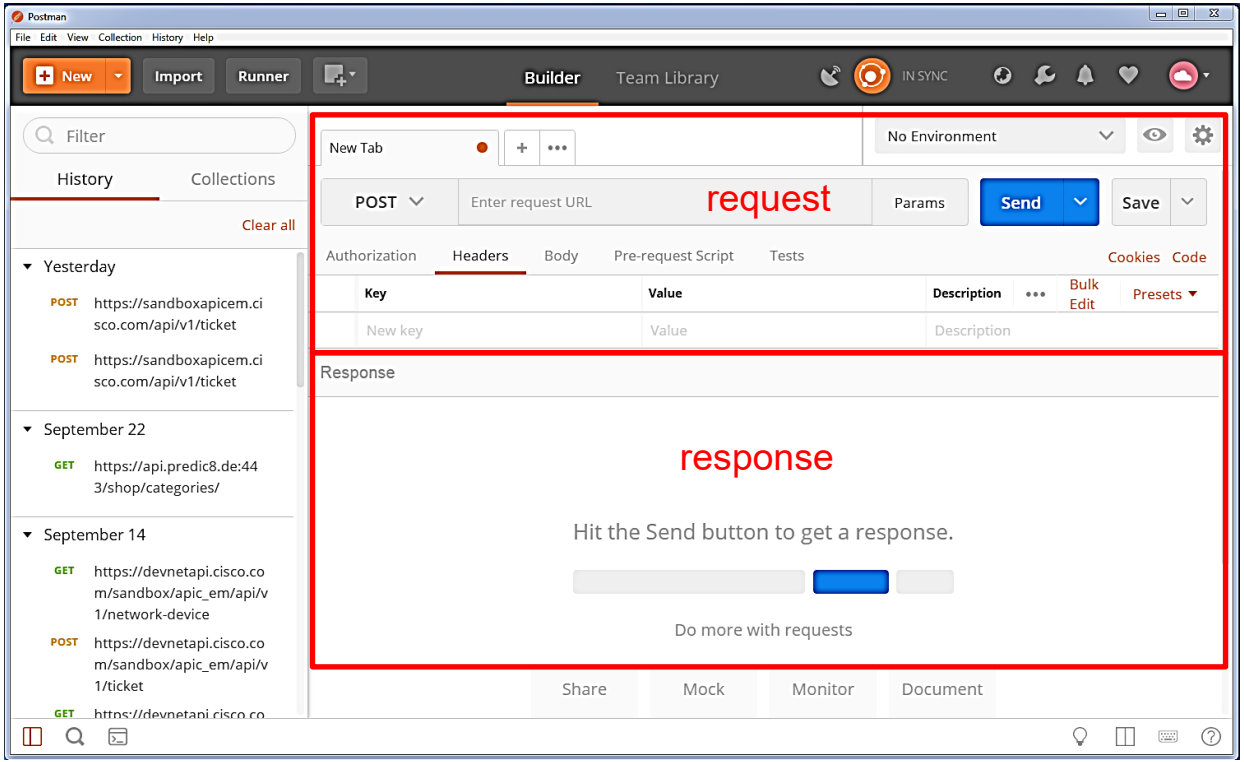
- An app for Windows, Mac OS, or Linux that provides an easy way to interact with REST APIs.
- Allows for headers to be easily constructed.
- Displays request status code and response data.
- Frequently used requests can be saved in tabs, history, or collections for reuse.

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.
- Under Request, set SSL Certificate Verification to "OFF"



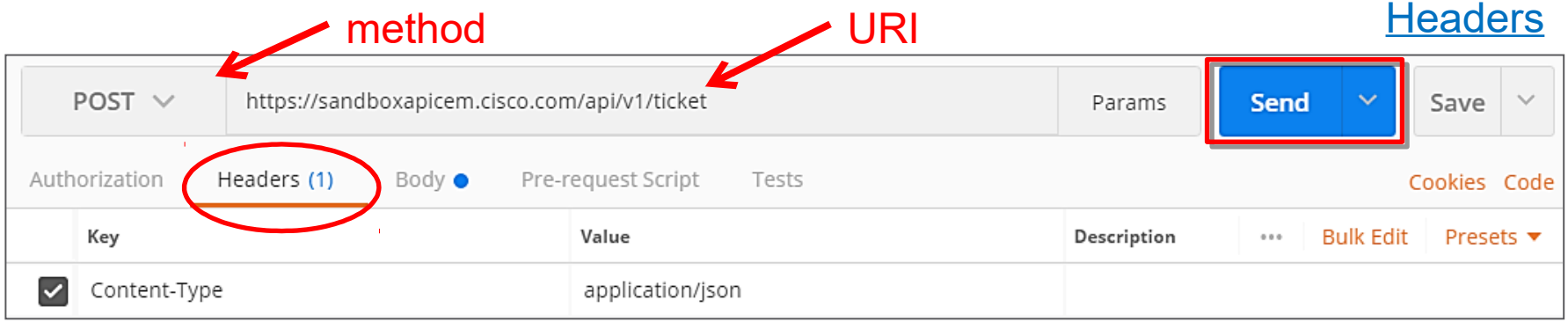
Postman Features



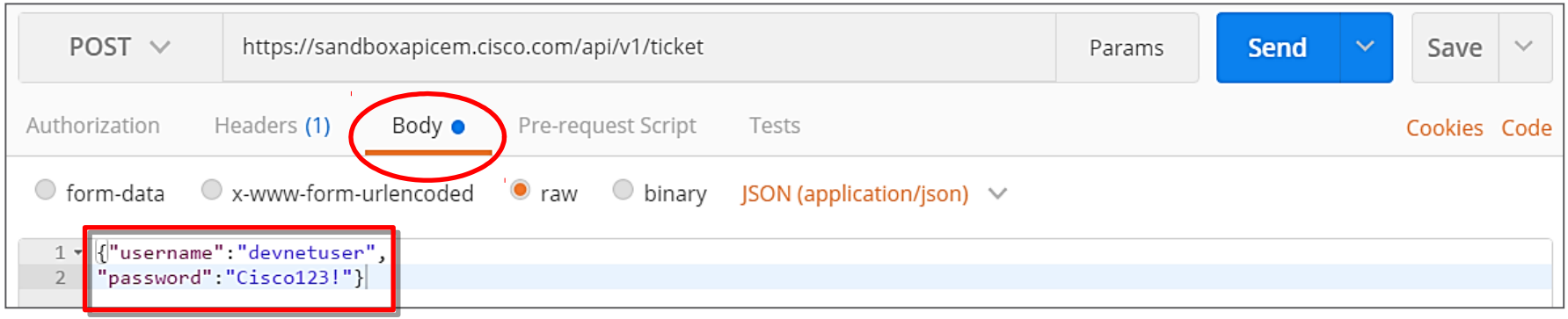
- History
- Tabs
- Collections
- Presets
- Code
- Environments
- Collaboration

Lab: Getting a Service Ticket with Python

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

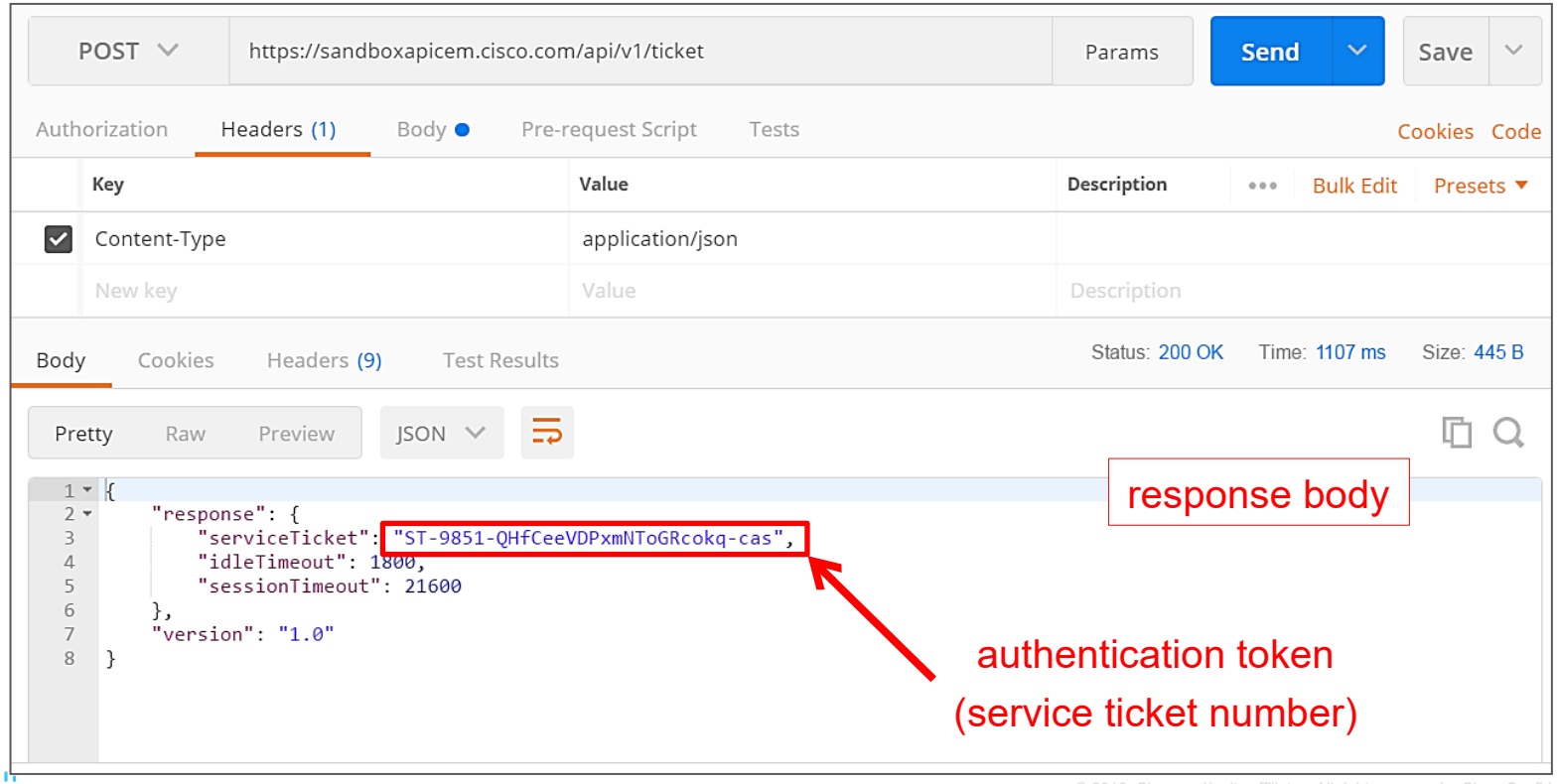


Body JSON: {"username": "devnetuser", "password": "Cisco123!"} Body



40

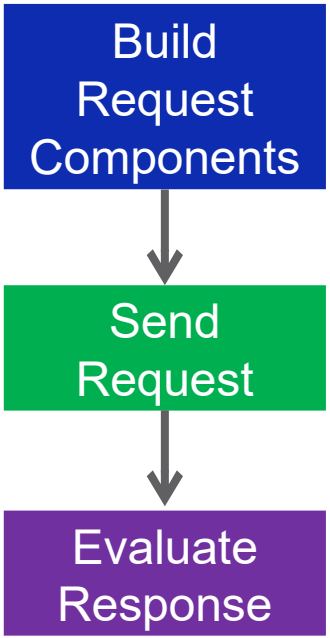
Lab: Getting a Service Ticket with Python
Part 1 Step 5b: View the Response



41

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

1. Open and run the **04_path_trace_sol.py** file.
2. From the list of devices, enter source and destination IP addresses.
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!

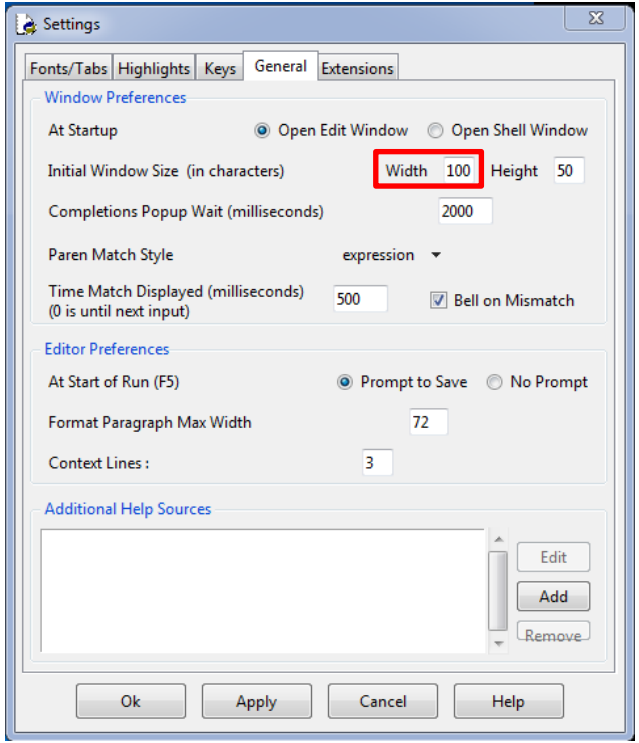
About Coding…

- 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.
  - **Paired symbols:** must match (open and close).
  - **Indentation:** standard indentation is four (or multiples of four) spaces, control structures additional 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 Environmen

```
"""
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": "devnetuser",
    "password": "Cisco123!"
}
```

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)
```

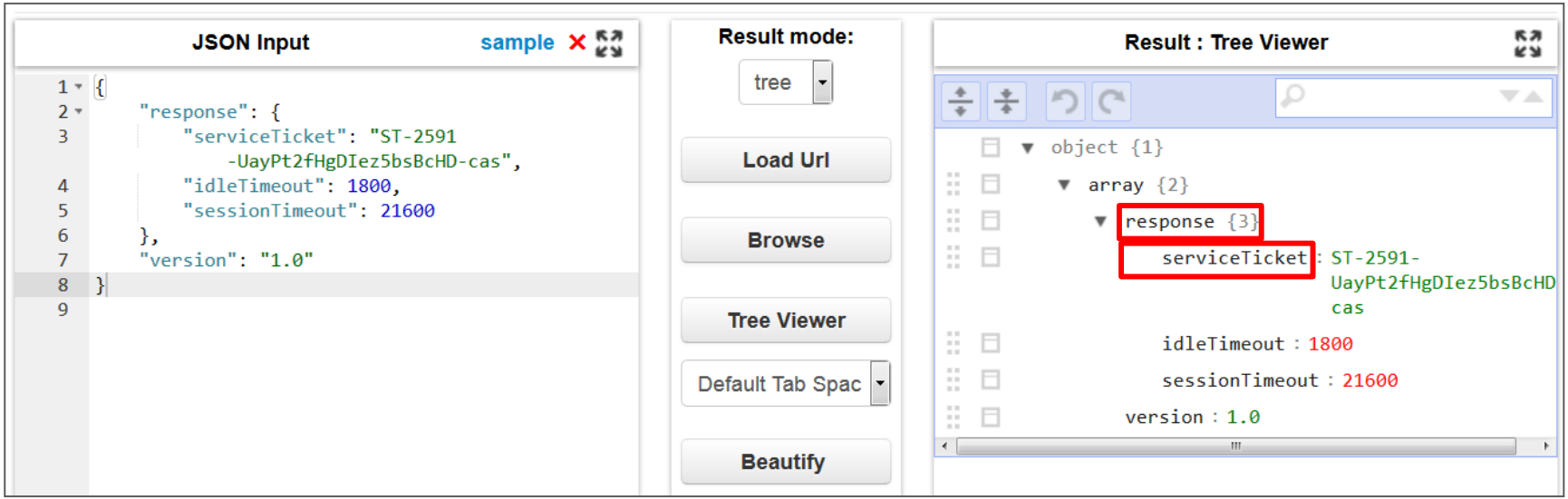
1. Create a Python object to hold the response to the request.
2. Provide the variables for the request to the **POST** method of the **requests** module.
3. **json.dumps()** encodes a Python object as JSON.

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




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

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

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

response_json = resp.json()

serviceTicket = response_json['response']['serviceTicket']

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

1. Create object with response code of request.
2. Display response code.
3. Decode the JSON **resp** variable into a python object and store in **response_json** object.
4. Extract the service ticket value from the object.
5. Display service ticket value.
6. 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**

Requirements for the function:

1. Defined with **`def get_ticket():`**
2. All subsequent lines of code indented an additional four spaces.
3. Function should return the service ticket number for use in other programs.

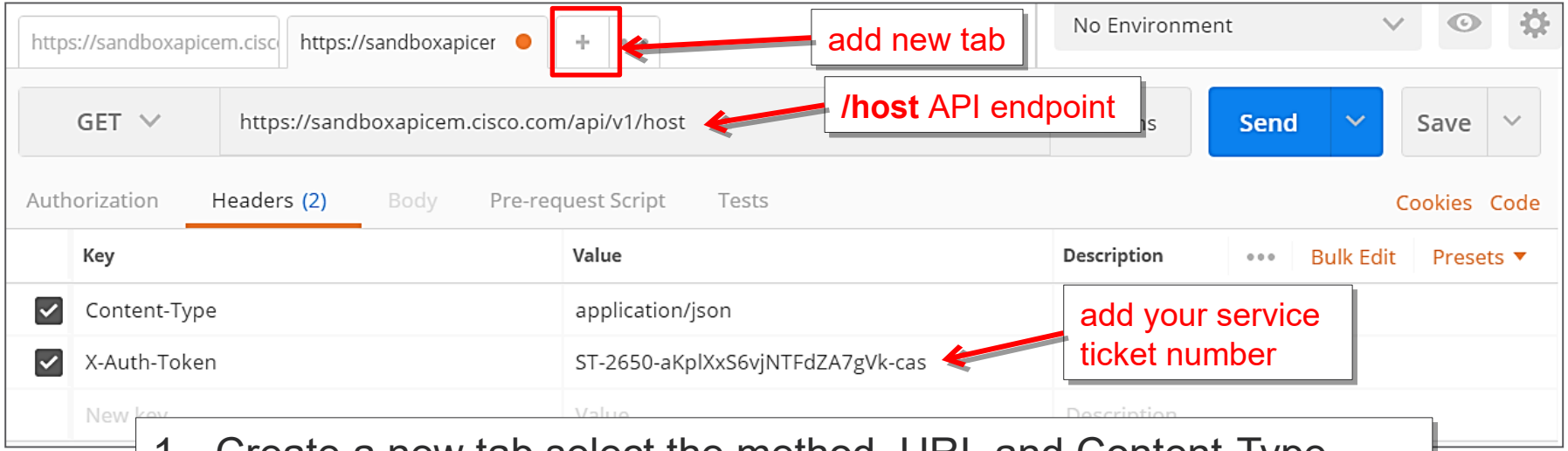
```
return serviceTicket
```

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.

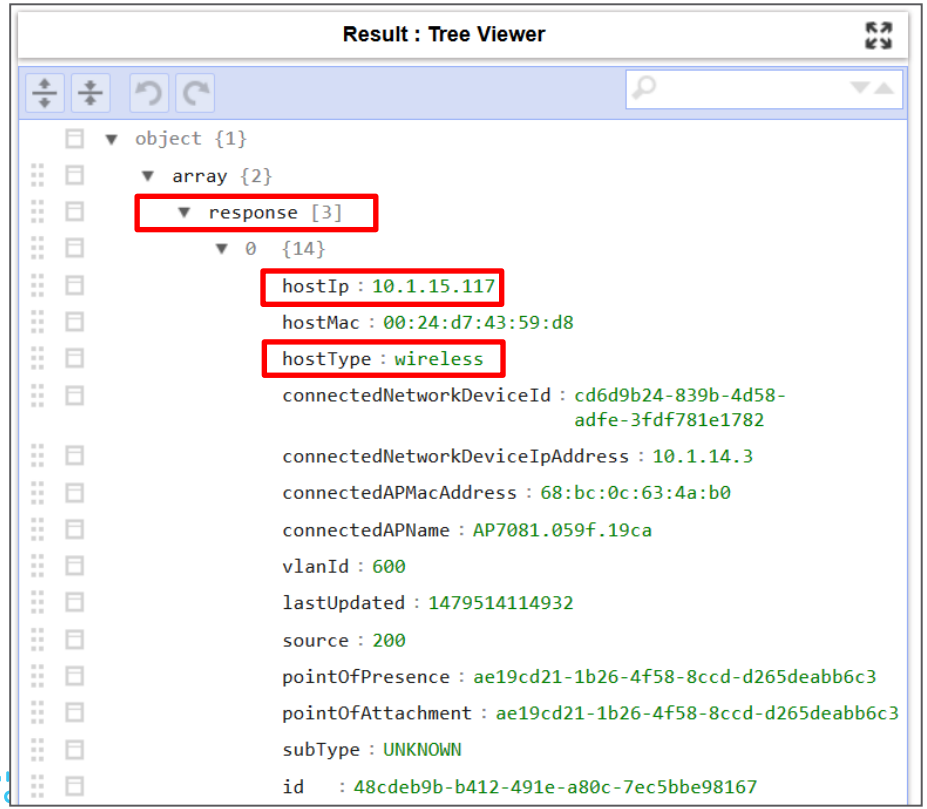
Lab - Create a Host Inventory in Python

Lab 2: Create a Host Inventory in Python
Part 1 Step 1: Setup the Postman Request



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



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 code as **print_hosts.py**

Lab: Create Host Inventory in Python

Part 2 Step 2: Build Request Components

```
api_url = "https://sandboxapicem.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 **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 as **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.
6. Save the file and test. Add the function **print_devices()** to your functions file.

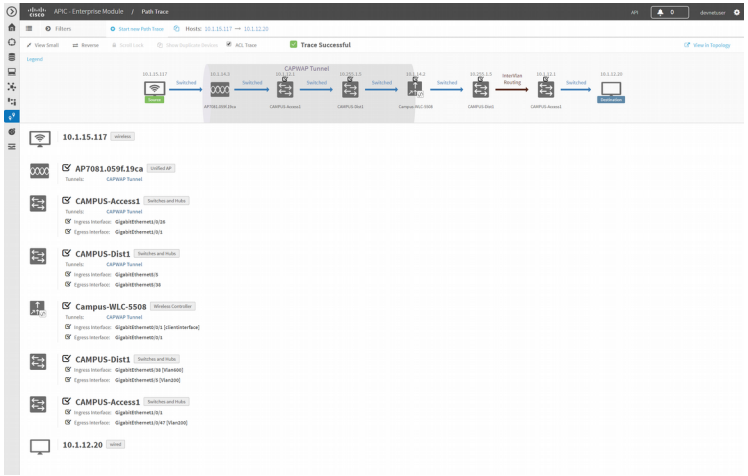
Lab - Using the APIC-EM Path Trace API

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

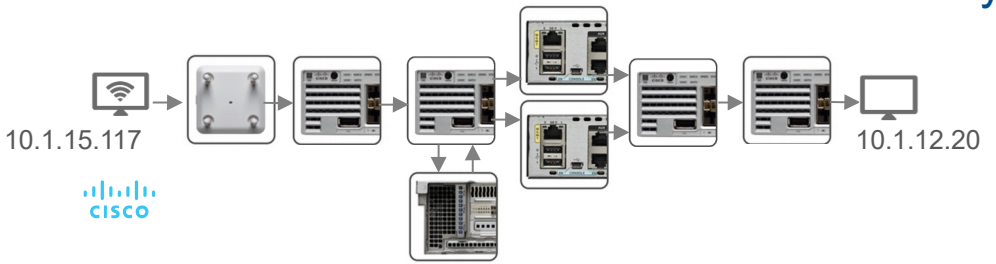


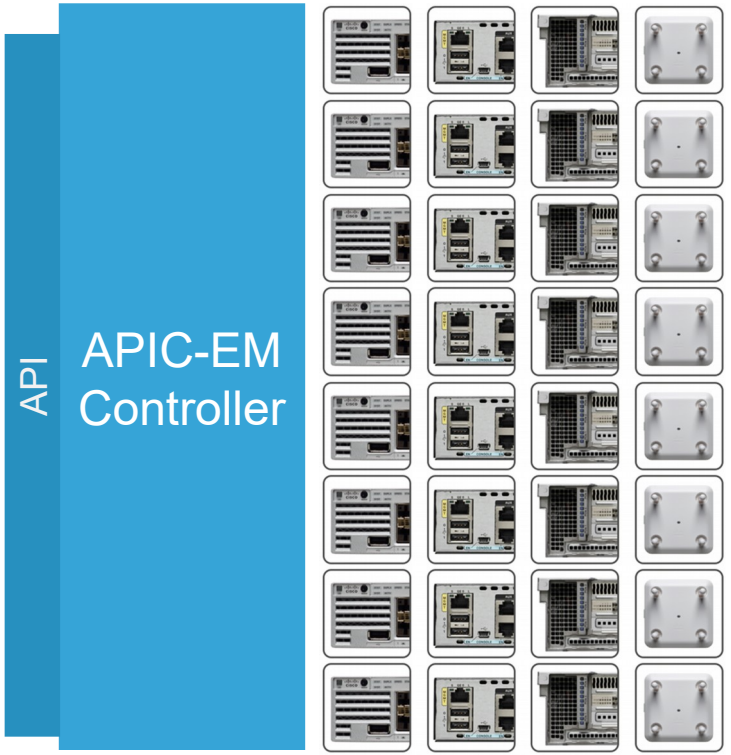
Use case:
How to optimize the Network Helpdesk team operations with the APIC-EM Path Trace?
E.g. user reports being unable to access from his PC the File server.

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 work
3. Is the **ID** work done? →
4. {JSON} ← The output is now ready!





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
#++++++++++++++++++++++++++++++++++++
```

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('List of hosts on the network: ')
# Add function to display hosts

print('List of devices on the network: ')
# Add function to display network devices

#++++++++++++++++++++++++++++++++++++
```

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 should 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+++++++++++++++
# Post request to initiate Path Trace
# Make the request. Construct the POST request with the json formatted path_data to the API
resp = 
# Inspect the return, get the Flow Analysis ID, put it into a variable
resp_json = resp.json()
flowAnalysisId = Assign the value of the flowAnalysisID key of resp_json.
#++++++++++++++++++++++++++++++++++++

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
check_url = [redacted]# Append the "/" + flowAnalyisId to the flow analysis
end point API URL that was created in Section 1
#++++++++++++++++++++++++++++++++++++
```

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 =              # 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

response_json["response"]["request"]["status"]

JSON Input — sample


```
{
    "response": {
        "request": {
            "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": [
            {
                "id": "48cdeb9b-b412-491e-a80c-7ec5bbe98167",
                "type": "wireless",
                "ip": "10.1.15.117",
                "linkInformationSource": "Switched"
            },
            {
                "id": "cd6d9b24-839b-4d58-adfe-3fdf781e1782",
                "name": "AP7081.059f.19ca",
                "type": "Unified AP",
                "ip": "10.1.14.3",
                "role": "ACCESS",
                "linkInformationSource": "Switched",
                "tunnels": [
```


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

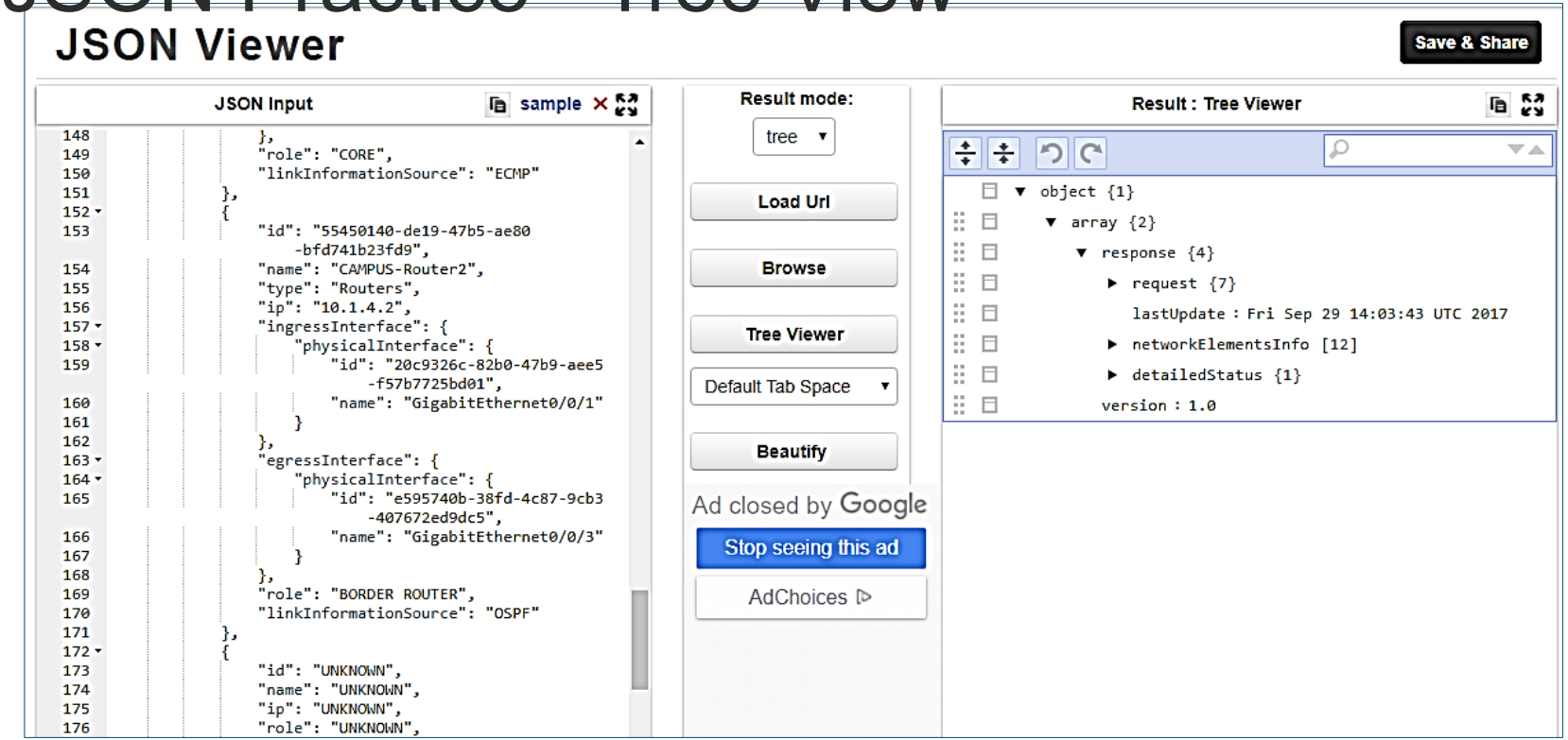
```
#============================
# Section 6. Display results
#============================

# 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



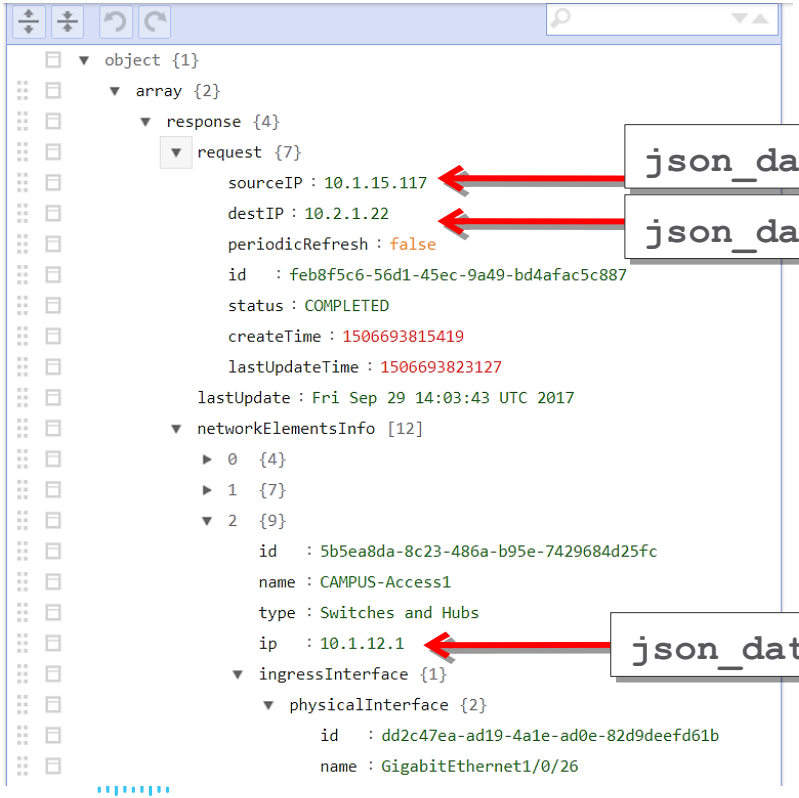
Lab: Using the APIC-EM Path Trace API JSON Practice - Load Variables

```
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



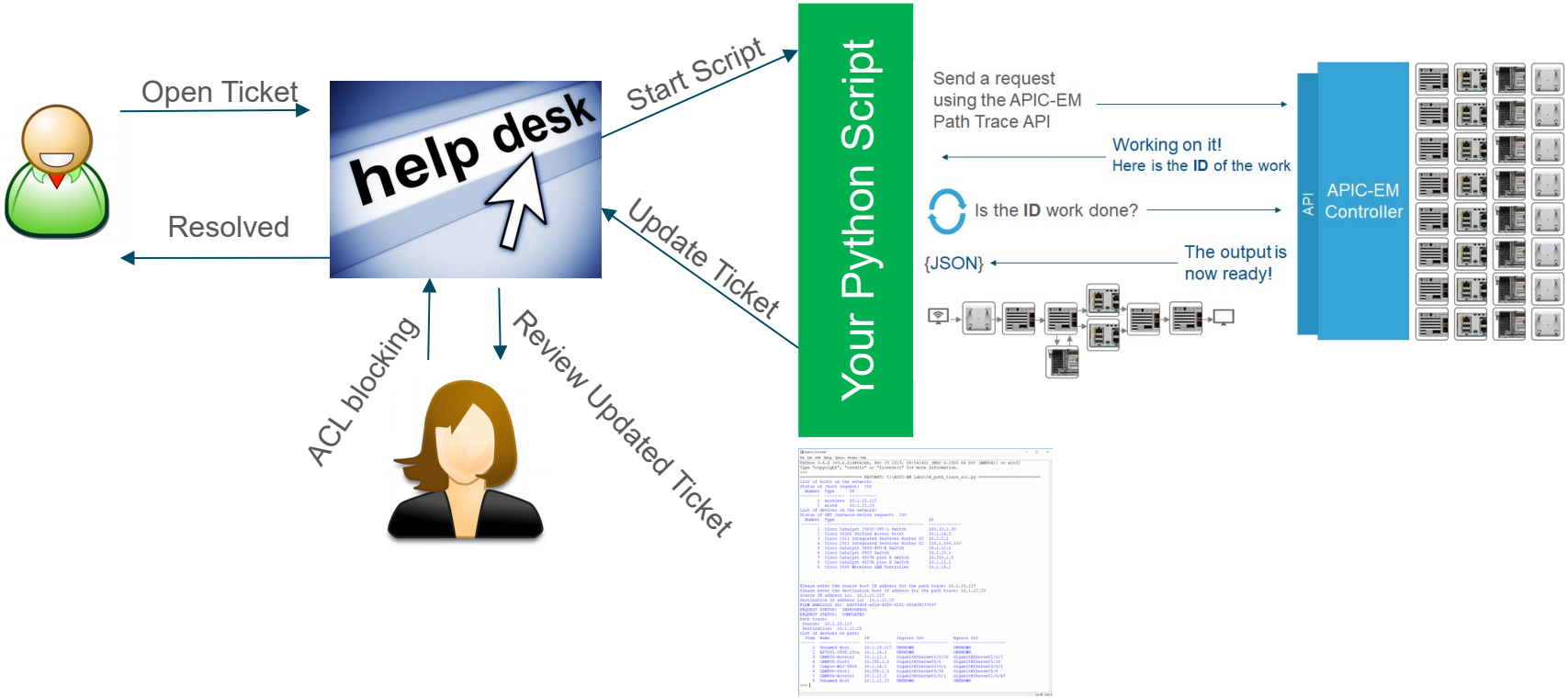

```
json_data['response']['request']['sourceIP')
```

```
json_data['response']['request']['destIP']
```

json_data holds the converted JSON reply from the Path Trace endpoint that is represented in the JSON Viewer tree.

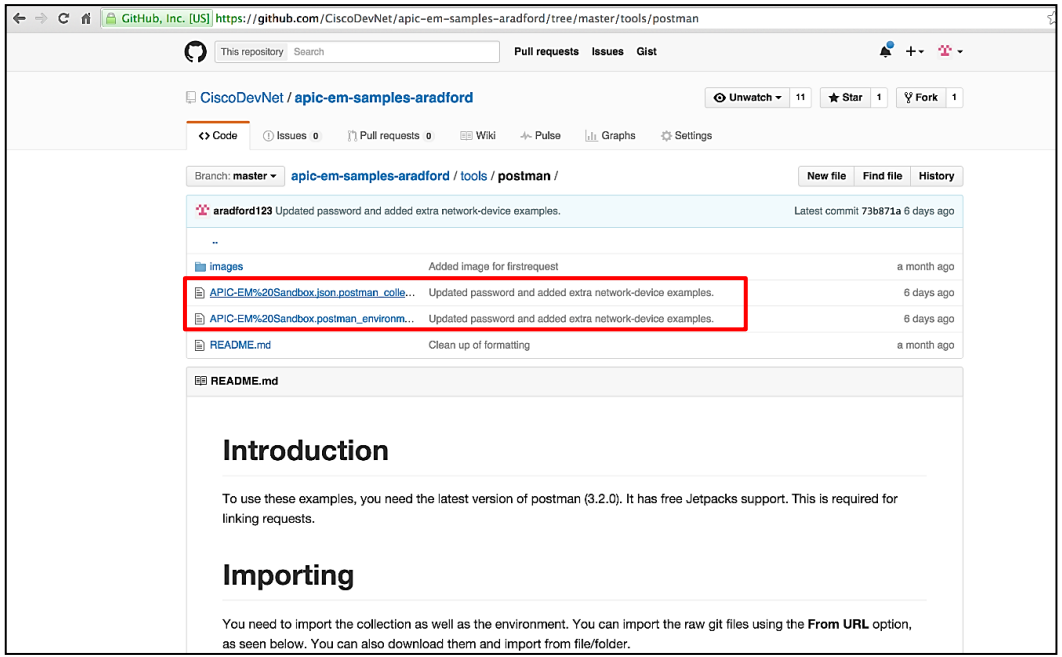
```
json_data['response']['networkElementsInfo'][2]['ip']
```

Lab summary



APIC-EM Tools DevNet GitHub Collection

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



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


CISCO