Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2025, gehouden op donderdag 27 maart 2025 jl. en georganiseerd door Hogeschool Windesheim). Bij elkaar zo'n 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op 18 maart 2027 in Arnhem en wordt georganiseerd door HAN University of Applied Sciences.

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

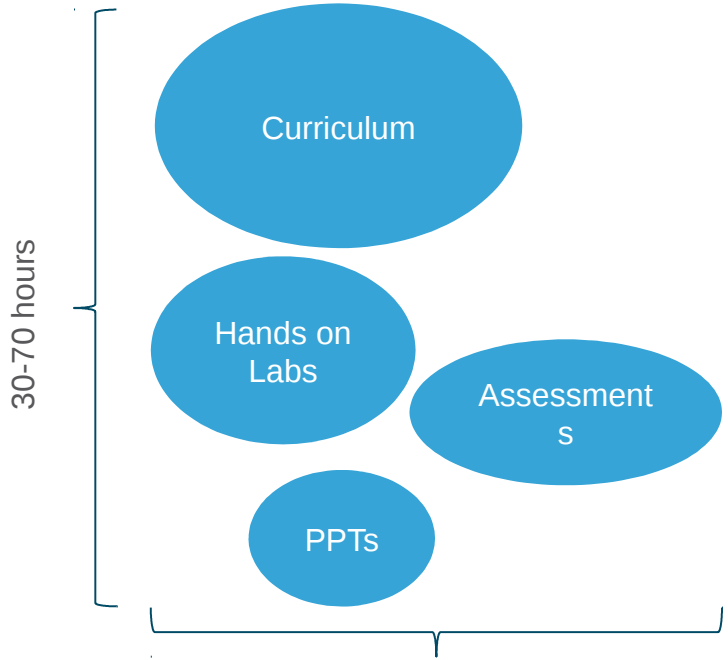
Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Emerging Technologies Workshop :
Network Programmability with Cisco APIC-EM

Networking Academy | DEVNET

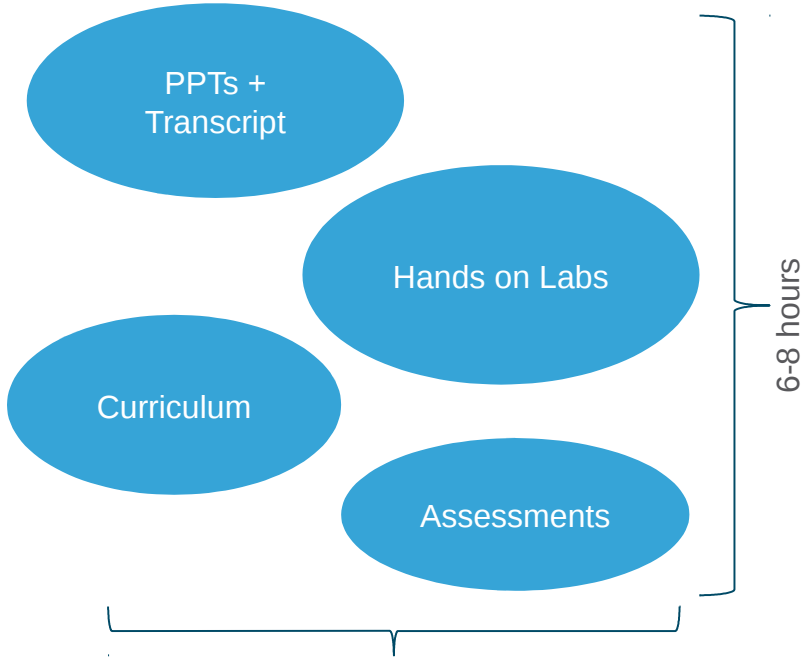
01/19/18

Traditional offering



Very comprehensive, end to end, up to carrier ready

Workshop offering



From Buzzwords to Hello World

1.0 Network Programmability

1.1 Devnet
1.2 SDN
1.3 The APIC-EM

2.0 Programming the APIC-EM REST API

2.1 REST
2.2 The APIC-EM API
2.3 Authentication
2.4 Lab 1 : Getting a Service Ticket with Python
2.5 Lab 2 : Create a host inventory in Python
2.6 Lab 3 : Create a network-device inventory in Python
2.7 Lab 4 : Path Trace

cisco Networking Academy | DEVNET

Workshop Objectives

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

- Explain how the Cisco APIC-EM enhances network management and performance 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!

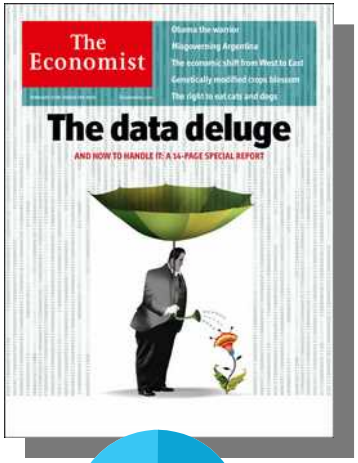
1.0 Network Programmability

1.1 Devnet

Why are we here?

Everything becomes **connected**

Everything becomes **software-based**



Everything generates **data**

Everything can be **automated**

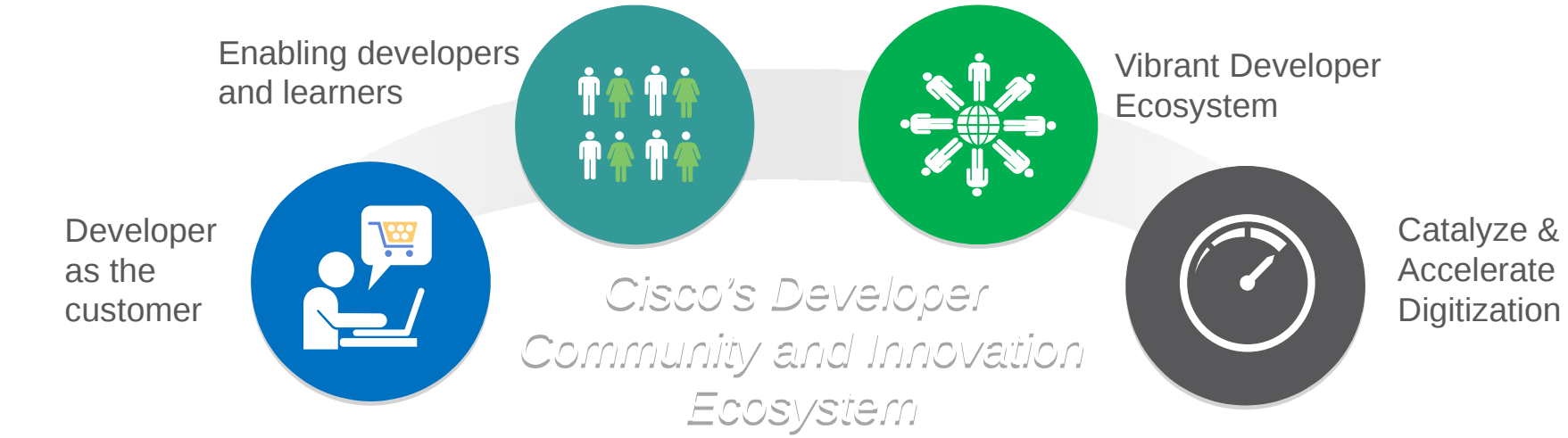
Everything needs to be **secured**

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

Introduction to DevNet Track

https://learninglabs.cisco.com/tracks/devnet-beginner

- Overview & DevNet Resources Beginner
  https://learninglabs.cisco.com/tracks/devnet-beginner/devnet-beginner-overview/01-intro-01-intro-to-devnet/step/1
- Intro to Coding Fundamentals
  https://learninglabs.cisco.com/tracks/devnet-beginner/fundamentals/intro-to-git/step/1
- Beginning APIs - Using Spark
  https://learninglabs.cisco.com/tracks/devnet-beginner/beginning-apis/00-prep-02-overview-of-rest-apis/step/1
- Network Programmability
  https://learninglabs.cisco.com/tracks/devnet-beginner/network-programmability/networking-101-the-basics/step/1

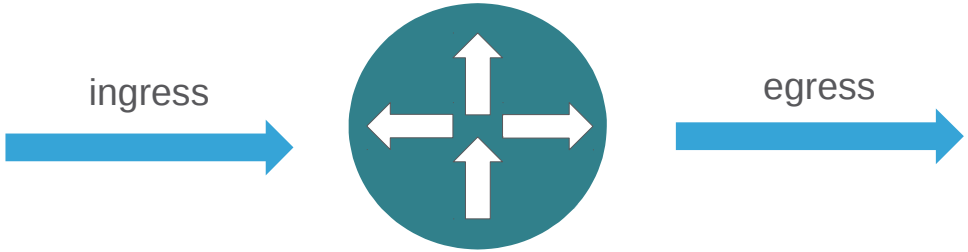
1.0 Network Programmability

1.2 SDN

SDN: Control Plane and Data Plane

Control Plane

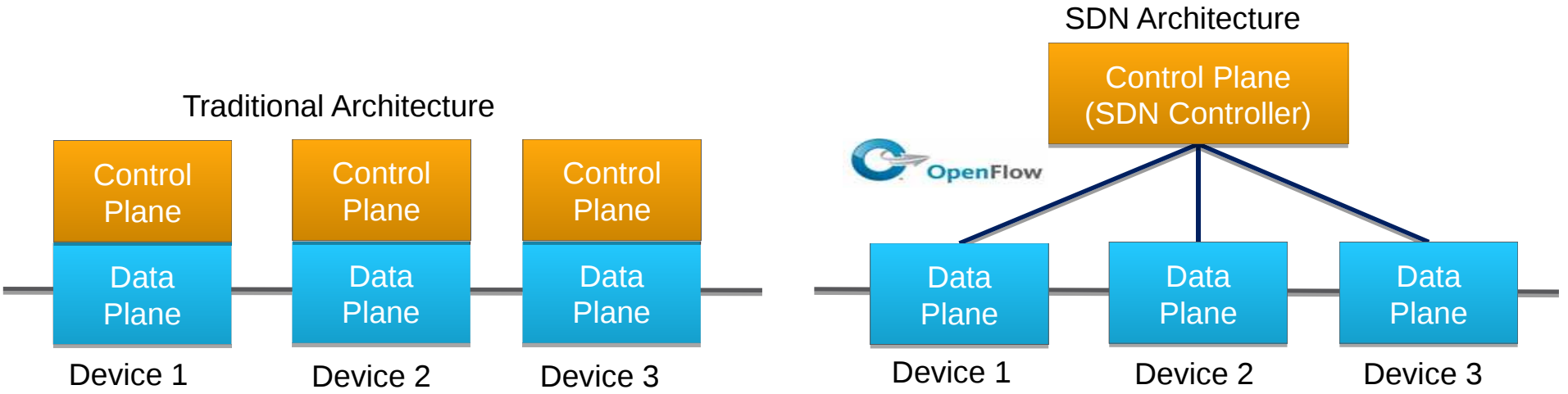
Hardware	Purpose	Example Processes
Device CPU	makes decisions about where traffic is sent	routing protocols, spanning tree, AAA, SNMP, CLI



Data Plane

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

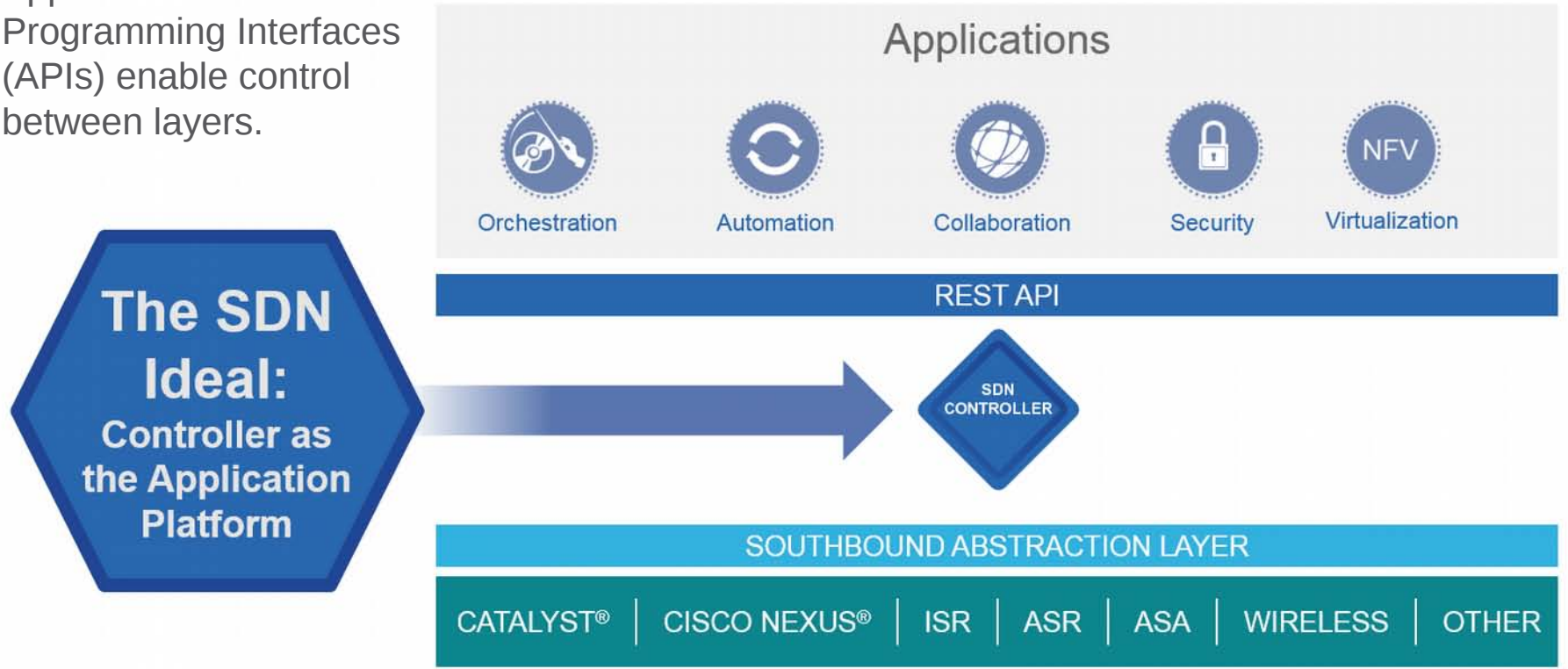
Traditional and SDN Architectures



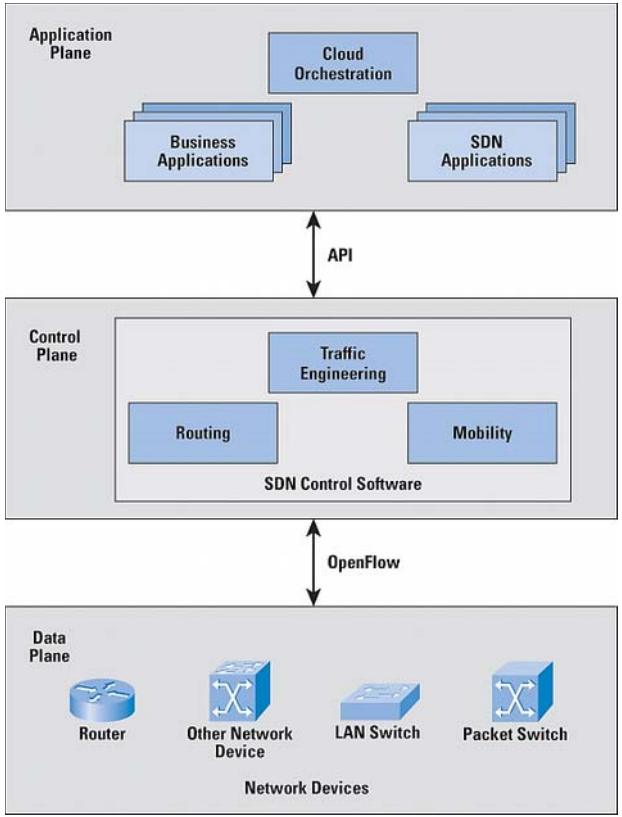
OpenFlow is a protocol between SDN controllers and network devices, as well as a specification of the logical structure of the network switch functions.

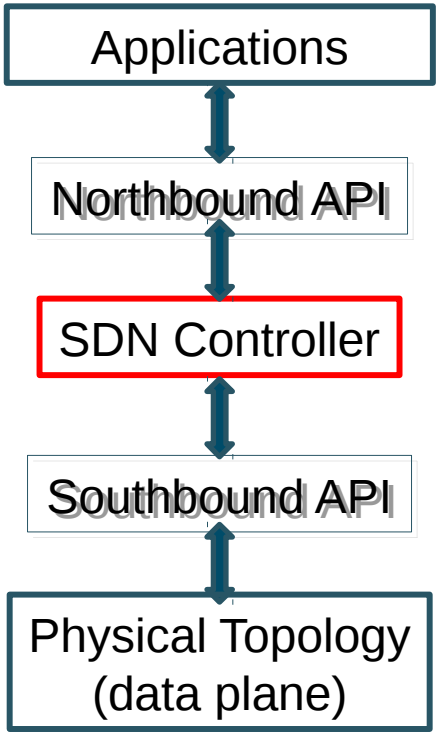
Network-wide Abstractions Simplify the Network

Application Programming Interfaces (APIs) enable control between layers.

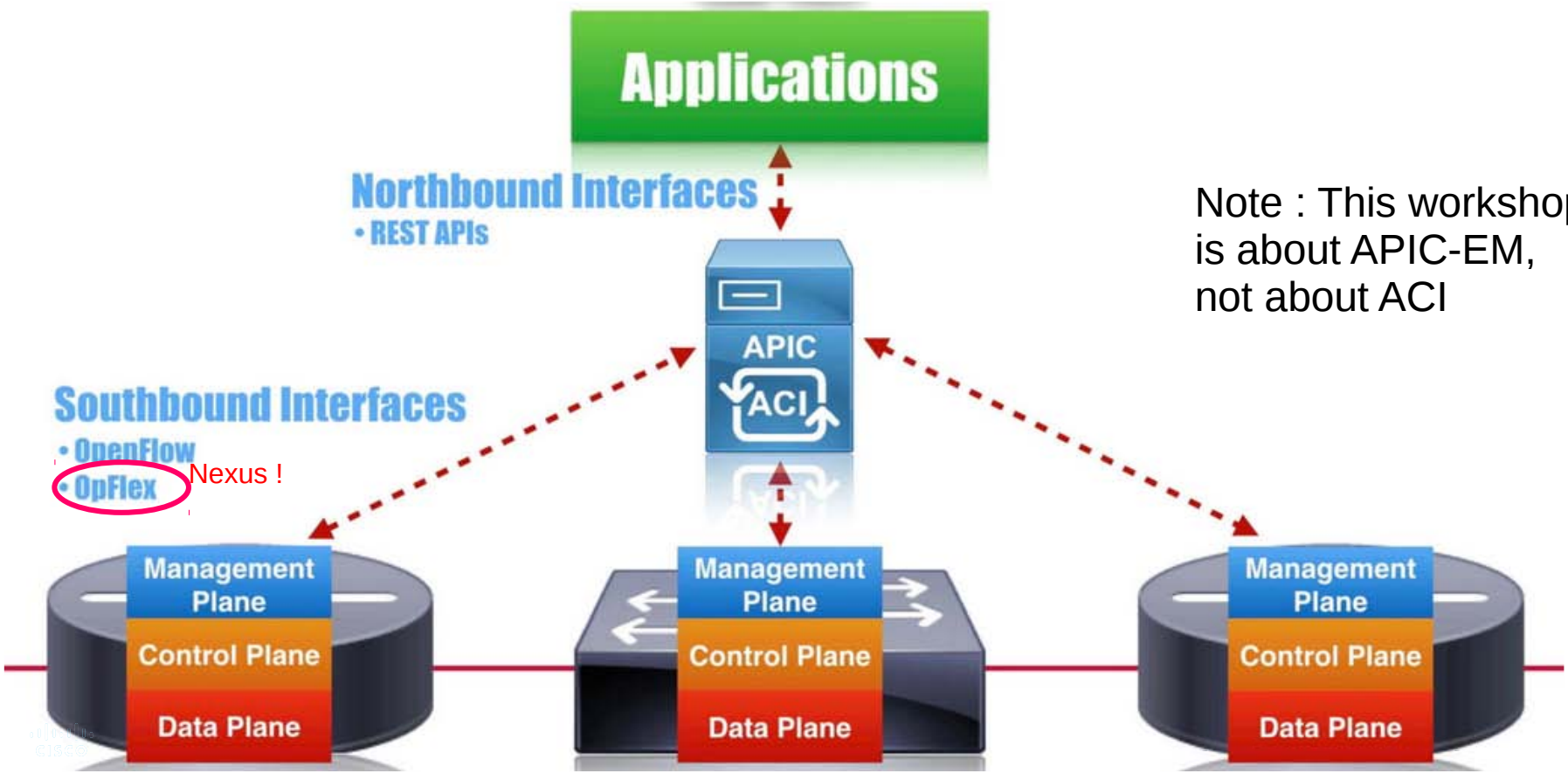


SDN Framework



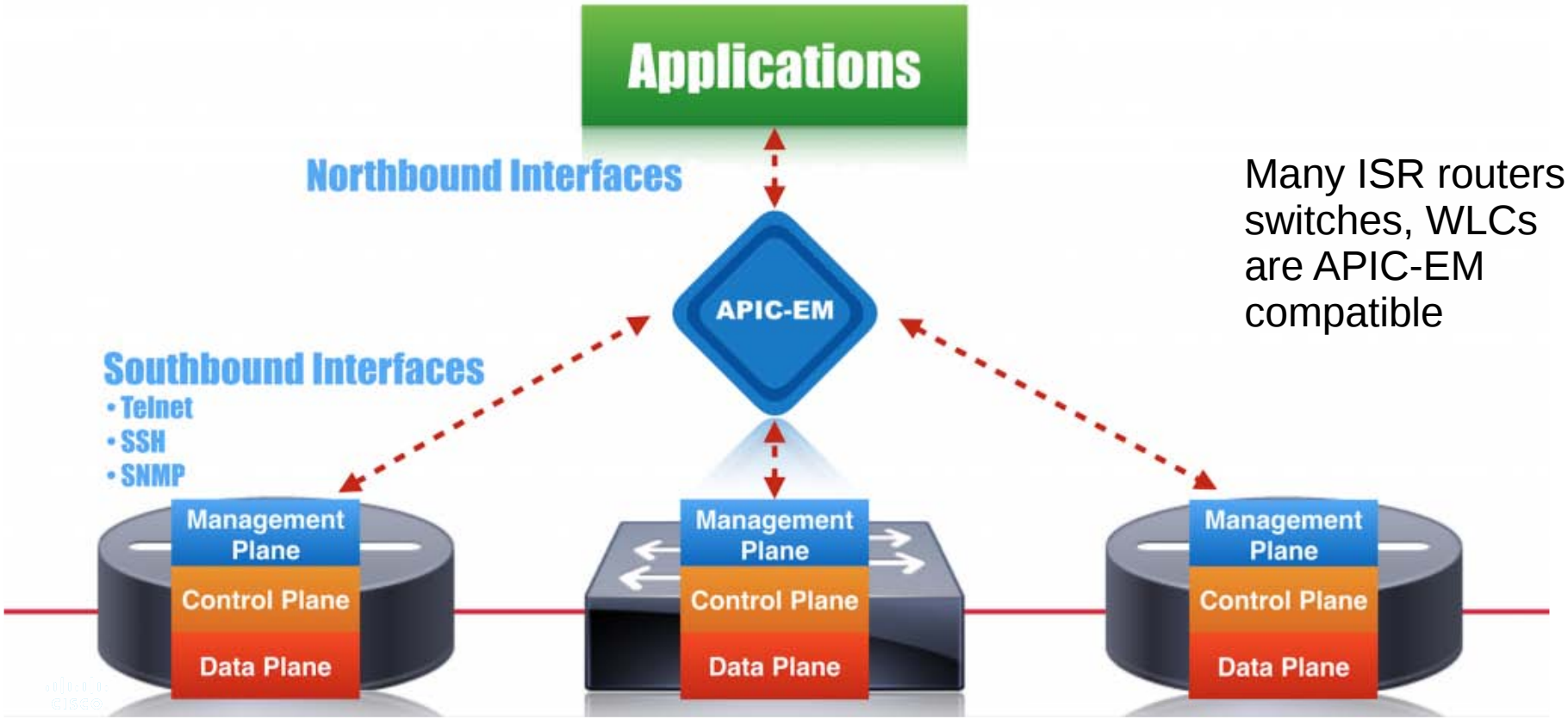


SDN : Cisco point of view in the Data Center : ACI



Note : This workshop is about APIC-EM, not about ACI

SDN : Cisco point of view in the campus : APIC-EM



Many ISR routers, switches, WLCs are APIC-EM compatible

1.0 Network Programmability

1.3 The APIC-EM

What is the APIC-EM?

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

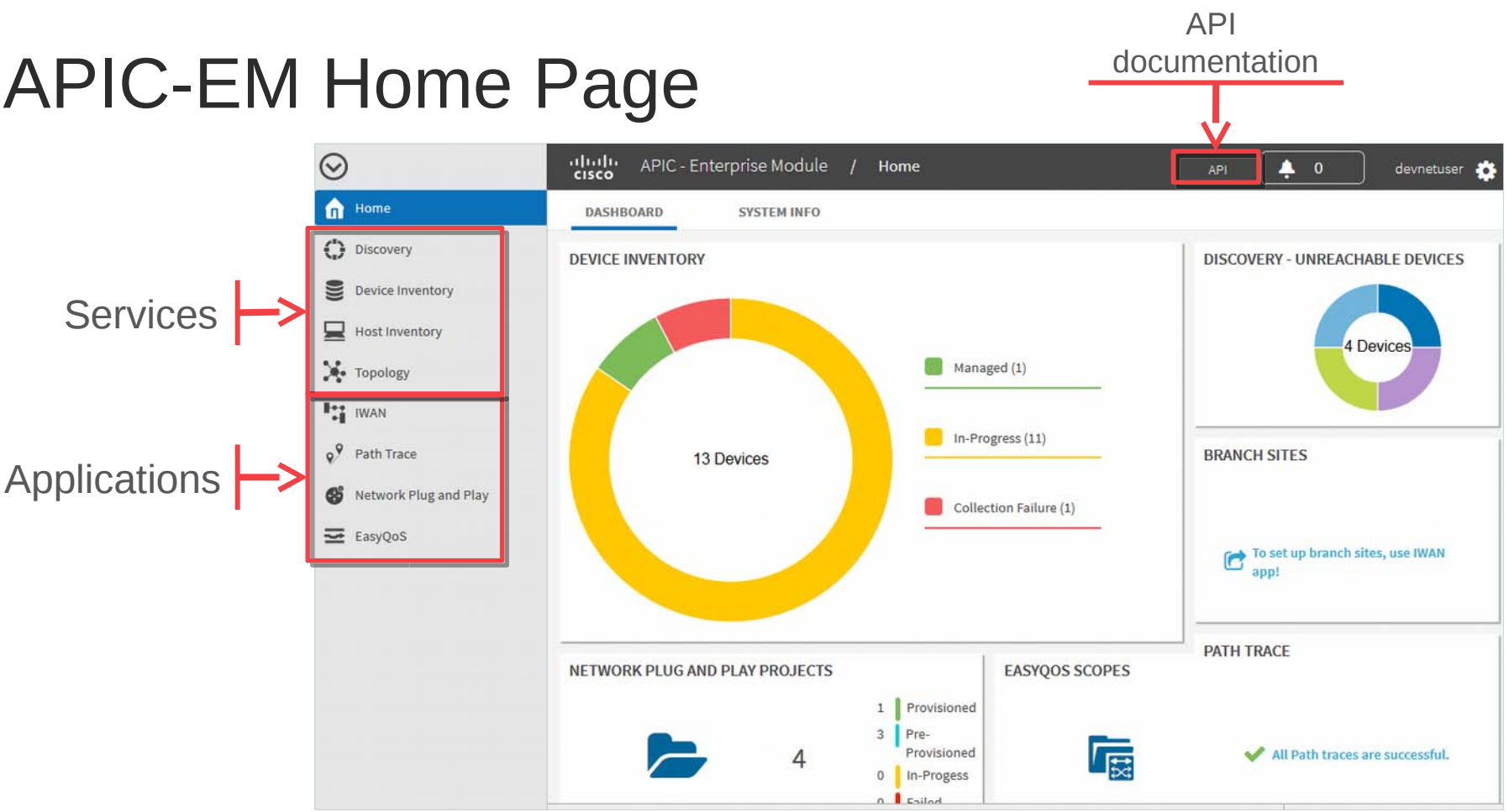
- A Software-Defined Networking (SDN) controller for enterprise networks
- A virtual, software-only, or physical appliance (>32GB RAM, 6 cores,...)
- Creates an intelligent, open, programmable network with open APIs
- Can transform business-intent policies into dynamic network configuration
- Provides a single point for network-wide automation and control
- The built in applications IWAN, Path Trace, Plug and Play, EasyQoS support enterprise routers, switches and Access Points
- All capabilities are exposed via a REST API

APIC-EM – Log in

https://sandboxapicem.cisco.com/

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

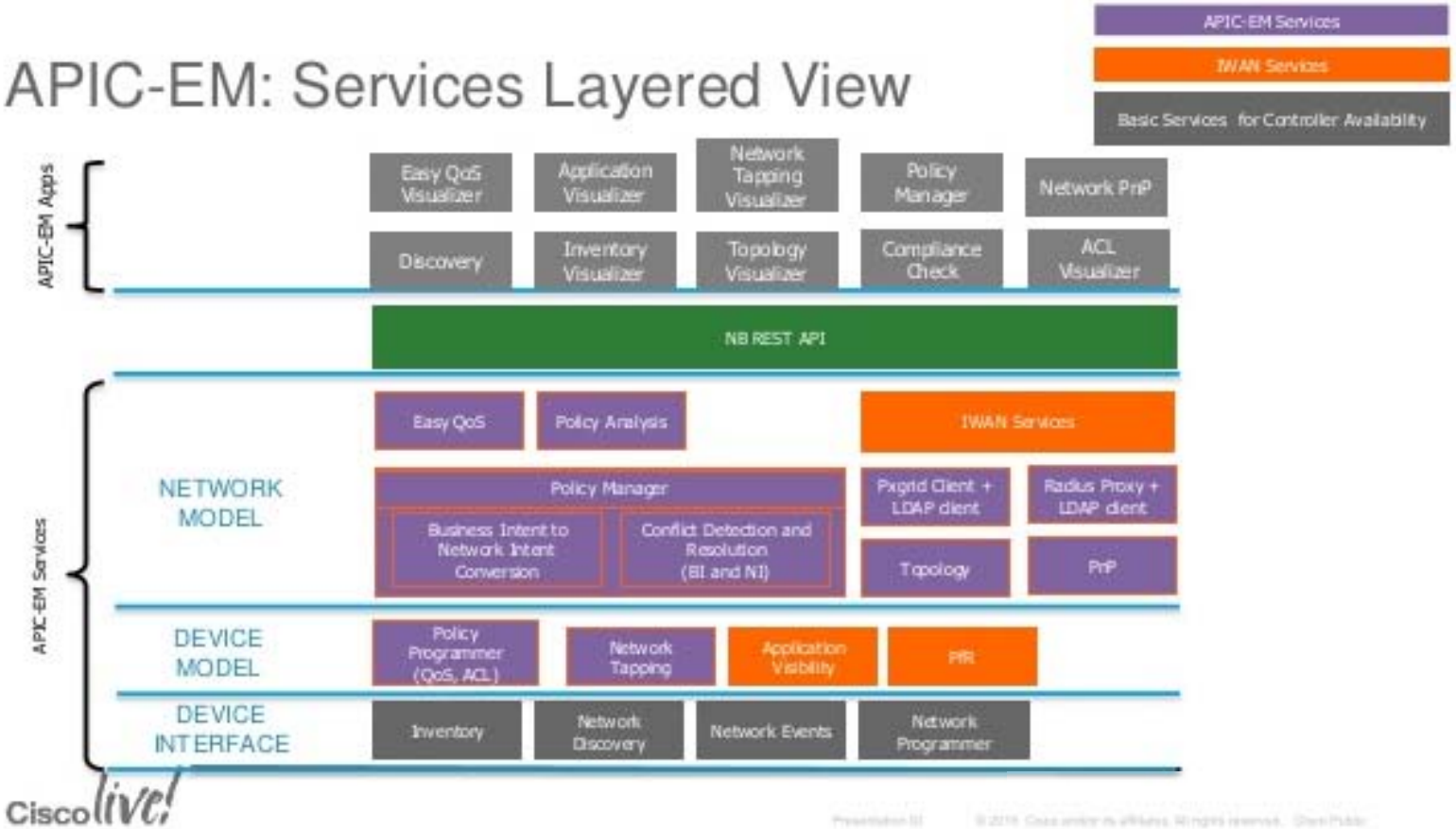
APIC-EM Home Page



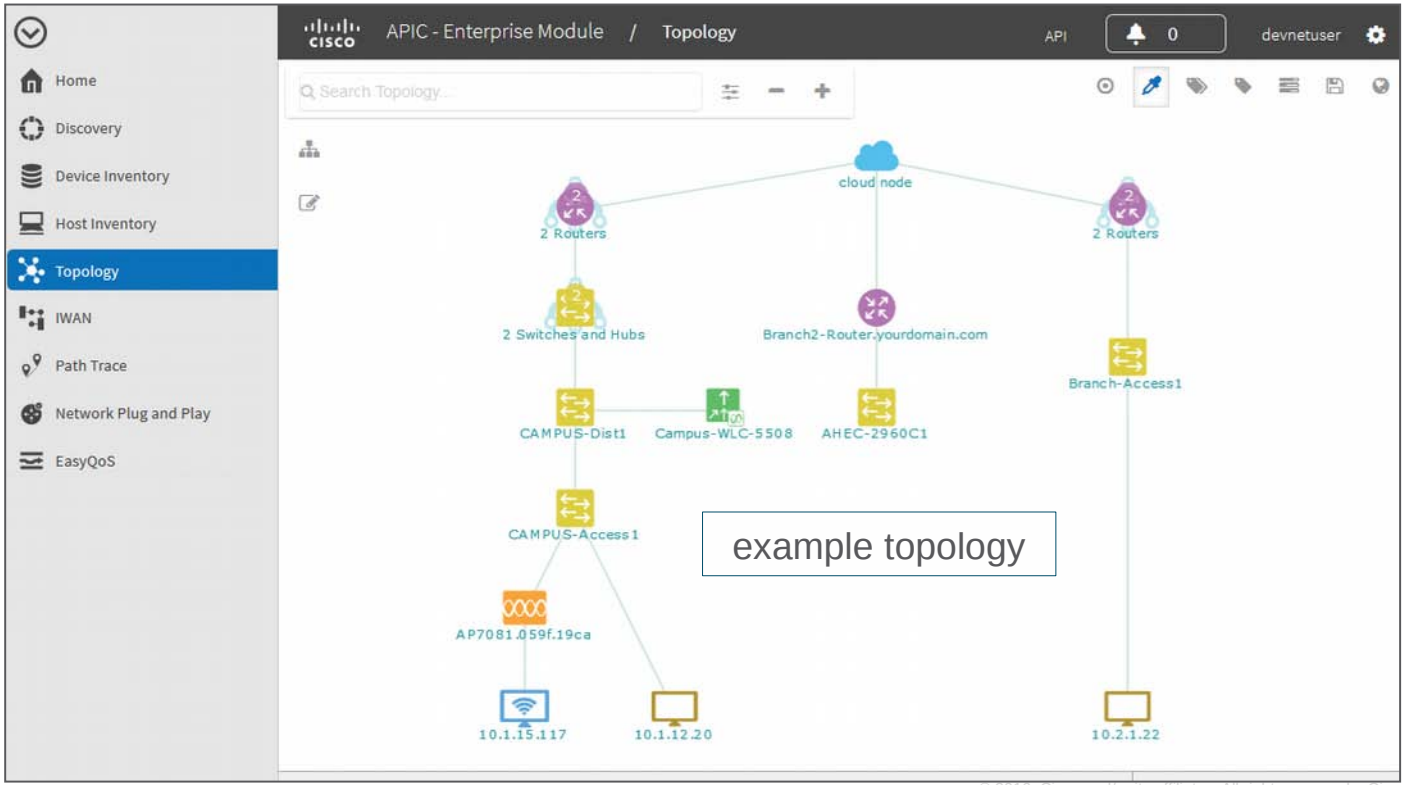
APIC-EM Applications

- **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 : What can it be used for ?



APIC-EM Topology Page



1.0 Network Programmability

1.1 Devnet
1.2 SDN
1.3 The APIC-EM

2.0 Programming the APIC-EM REST API

2.1 REST
2.2 The APIC-EM API
2.3 Authentication
2.4 Lab 1 : Getting a Service Ticket with Python
2.5 Lab 2 : Create a host inventory in Python
2.6 Lab 3 : Create a network-device inventory in Python
2.7 Lab 4 : Path Trace

2.0 Programming the APIC-EM REST API

2.1 REST

What is a web service?

- A web service is a way for two systems to communicate through a defined interface. Expl of web services : REST (Representational State Transfer) and SOAP (Simple Object Access Protocol)
- REST is an architecture style for designing networked applications.
- In REST, HTTP is used to communicate between 2 machines
- REST is a lightweight alternative to RPC, SOAP, Corba, …
- Example :
  GET http://www.acme.com/phonebook/UserDetails/12345

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

What is so great about REST*?

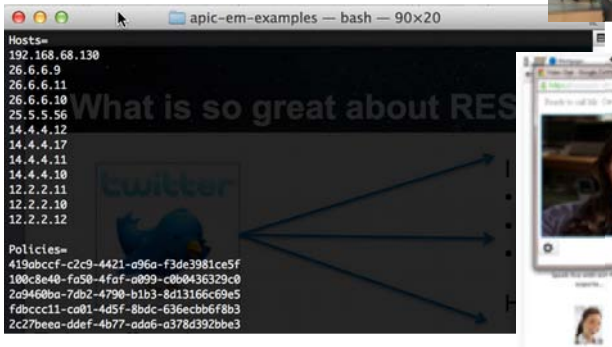
Cisco APIC-EM REST APIs

- Hosts
- Devices
- Users
- + more

How does this work?

Easy to use:

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

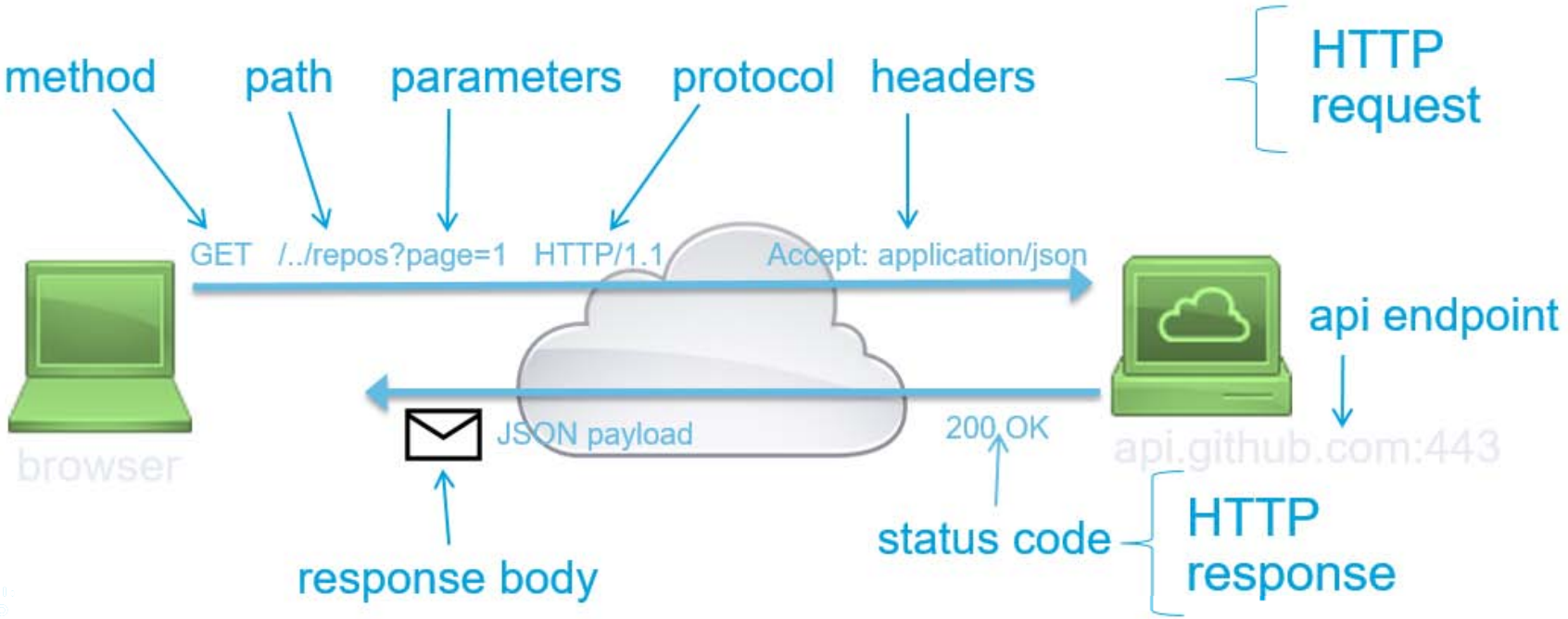


*representational state transfer (REST).

27

Anatomy of a REST API query

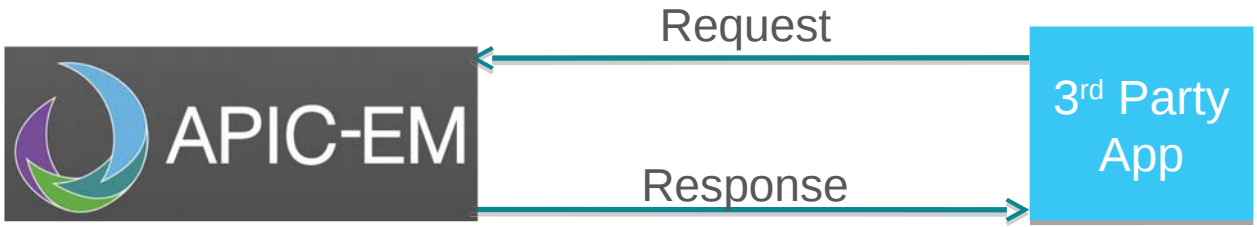
URL: https://api.github.com/users/CiscoDevNet/repos?page=1&per_page=2

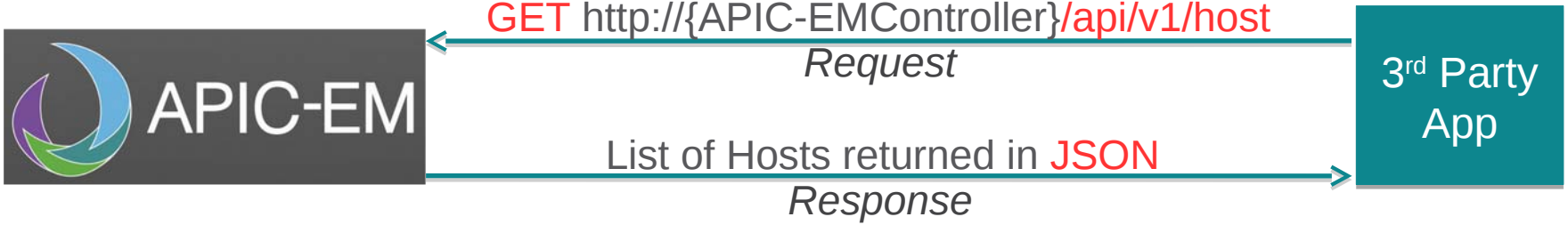


2.0 Programming the APIC-EM REST API

2.2 The APIC-EM API

How does this work?





Anatomy of a REST Request

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

Method

- GET, POST, PUT, DELETE

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

31

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

32

JSON and XML

JSON

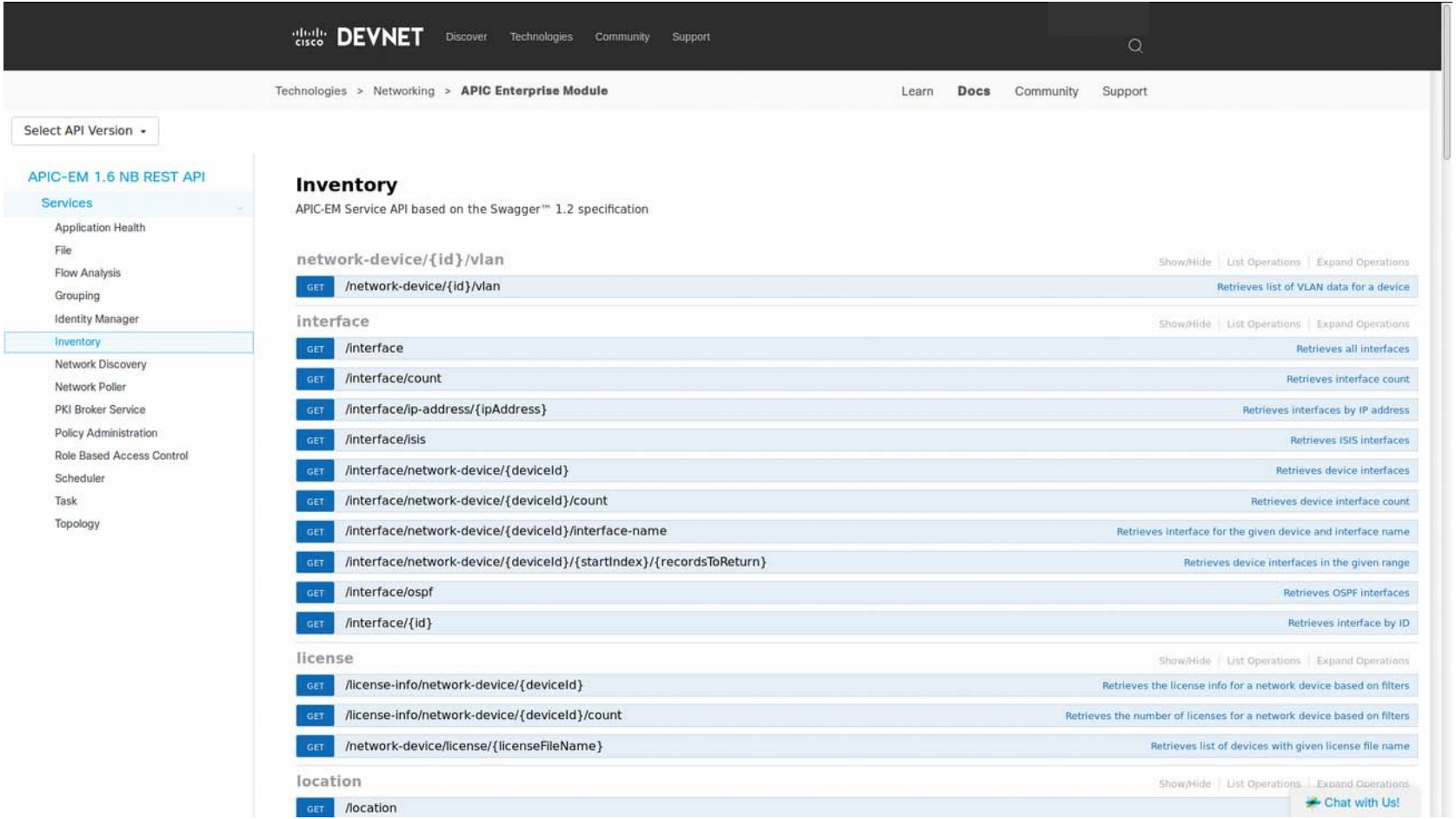
sample


```
{
    "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"
```

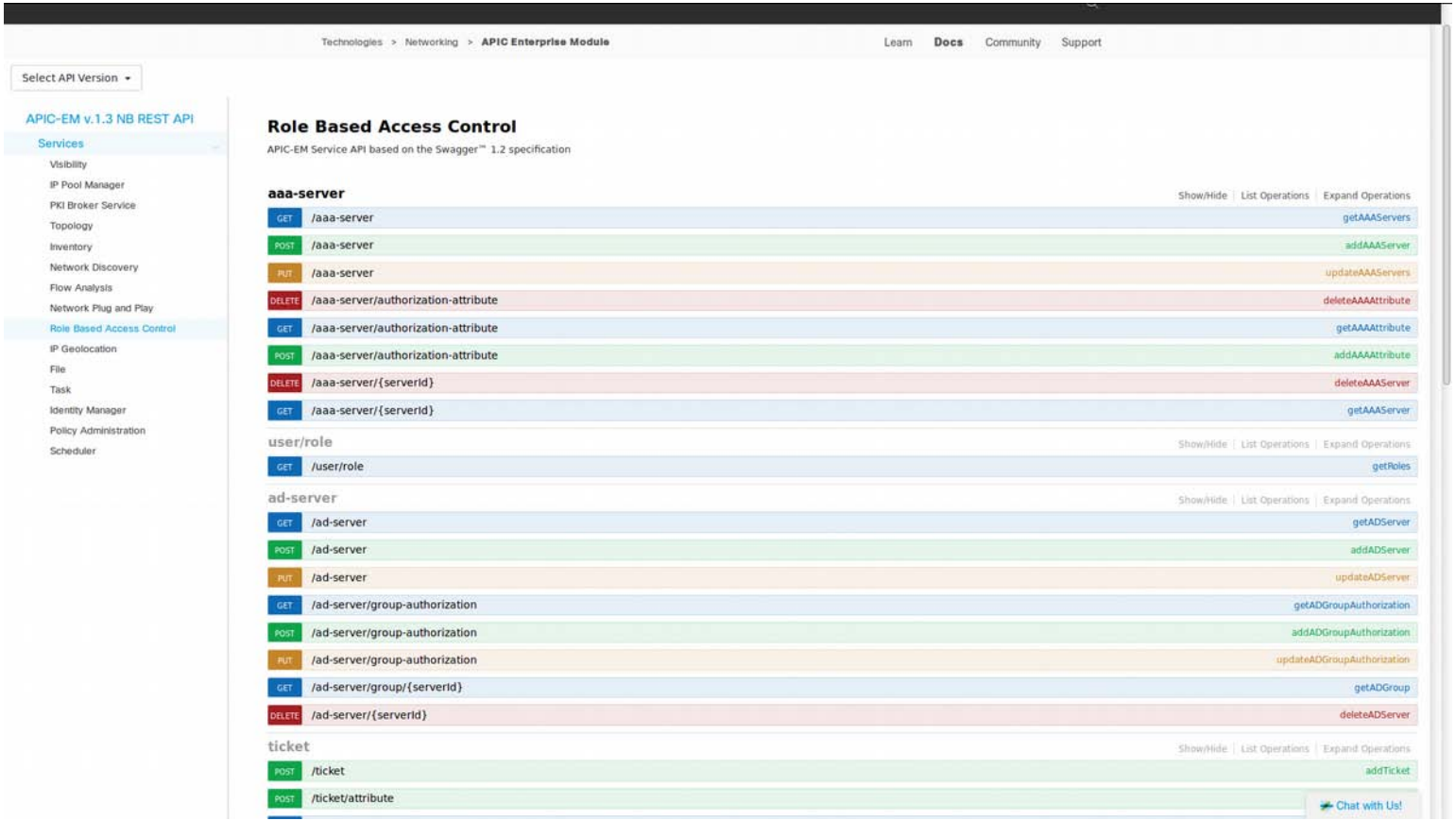

XML


```
<?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>
```

APIC-EM Documentation



APIC-EM Documentation



2.0 Programming the APIC-EM REST API

2.3 Authentication

What about authentication?

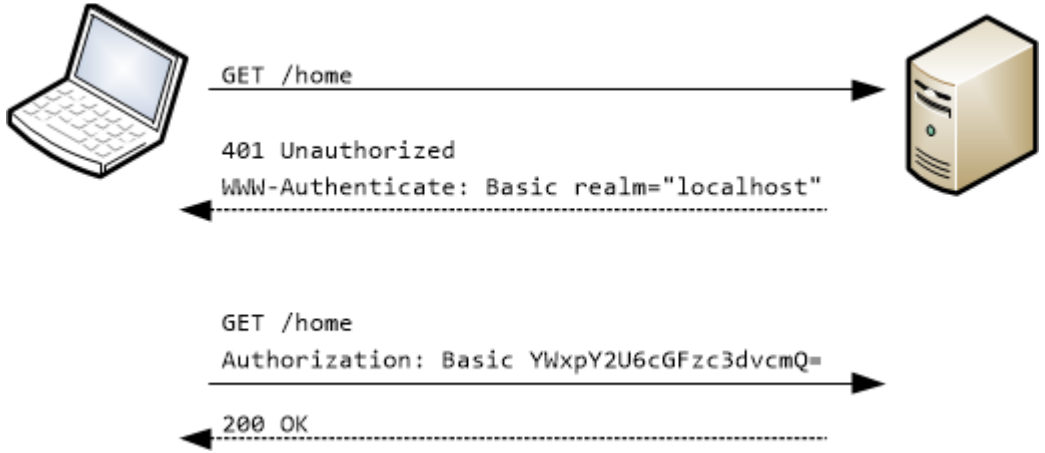
- **Basic HTTP:** The username and password are passed to the server in an encoded string.
- **Token:** A token is created and passed with each API call, but there is no session management and tracking of clients which simplifies interaction between the server and client.
- **OAuth:** Open standard for 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.

APIC-EM uses **Token** for authentication management. The APIC-EM calls this token a service ticket.

37

Basic HTTP

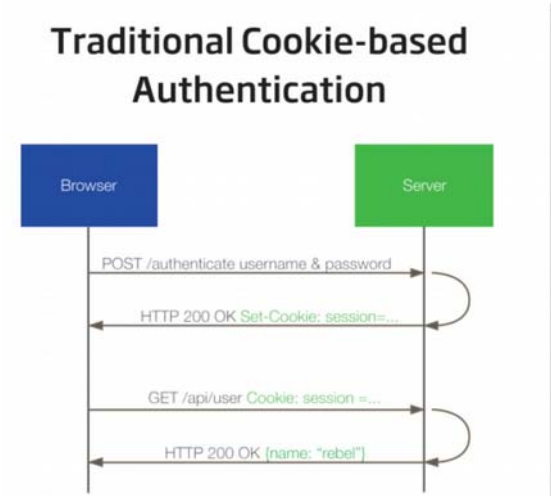
- **Basic HTTP:** The username and password are passed to the server in an encoded string. The server must keep track of this session – not scalable...



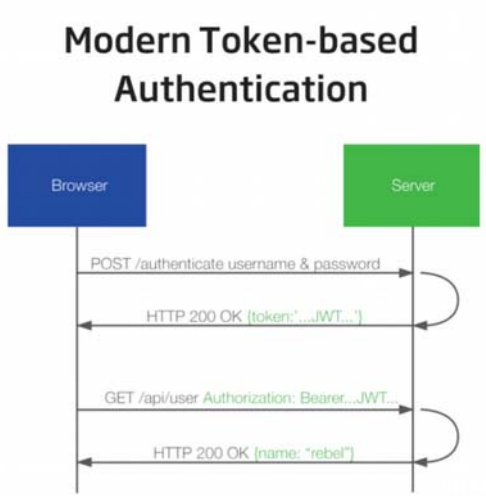
38

Token based authentication

- Stateless – no need to keep track of every user
- Token must be passed in every request from the client
- Token will be placed in the http header



Stateful : session-id must be kept on both ends

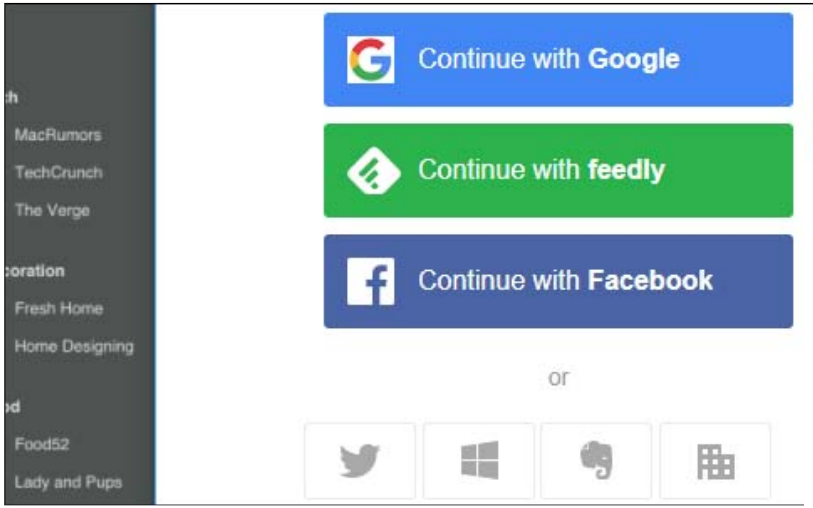


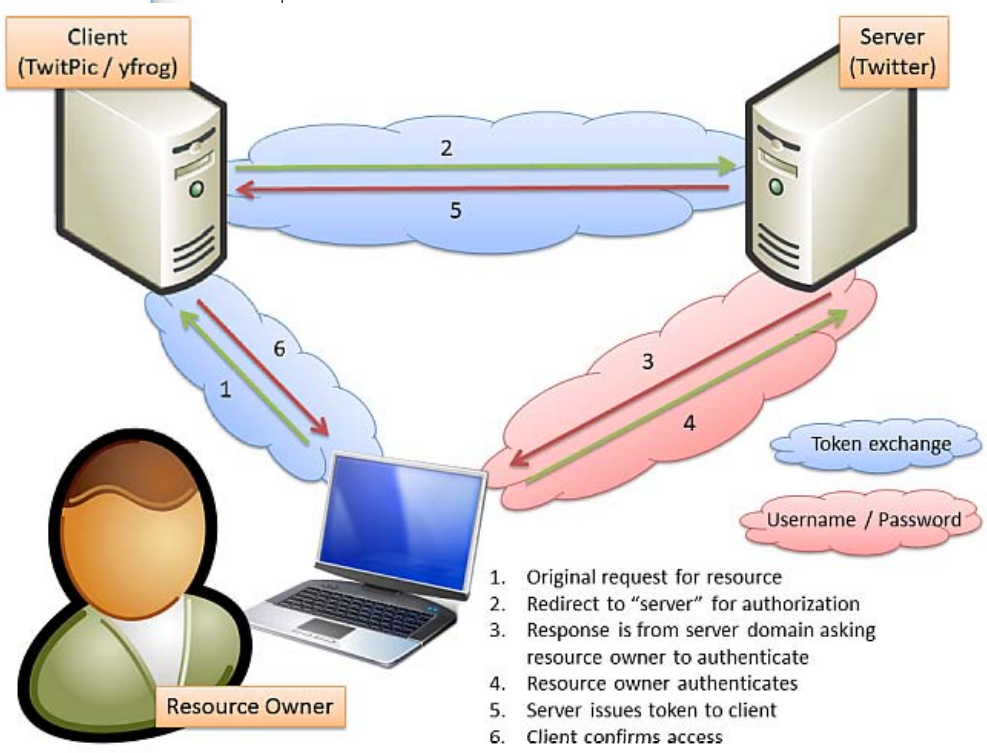
Stateless : token is signed by server and checked at each request

Analogy with conference badge...

39

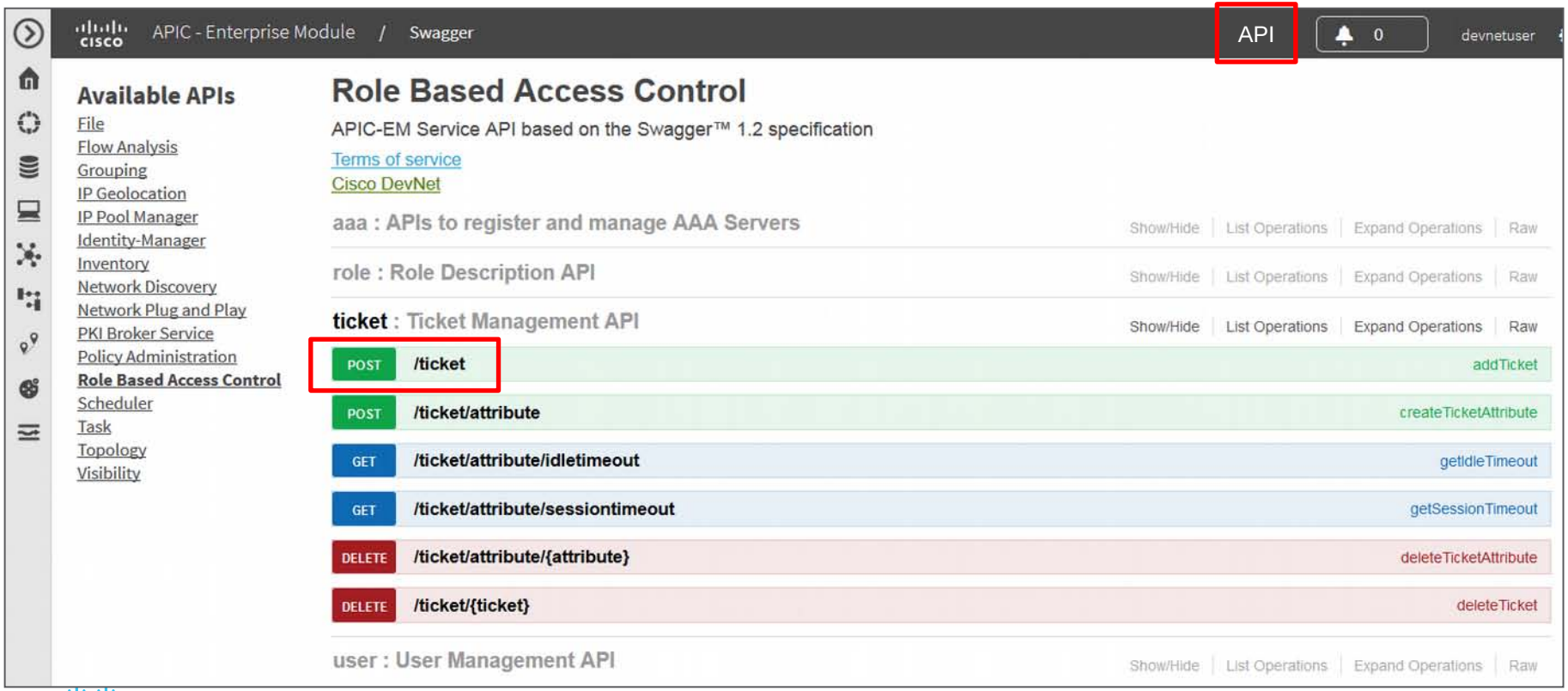
OAUTH





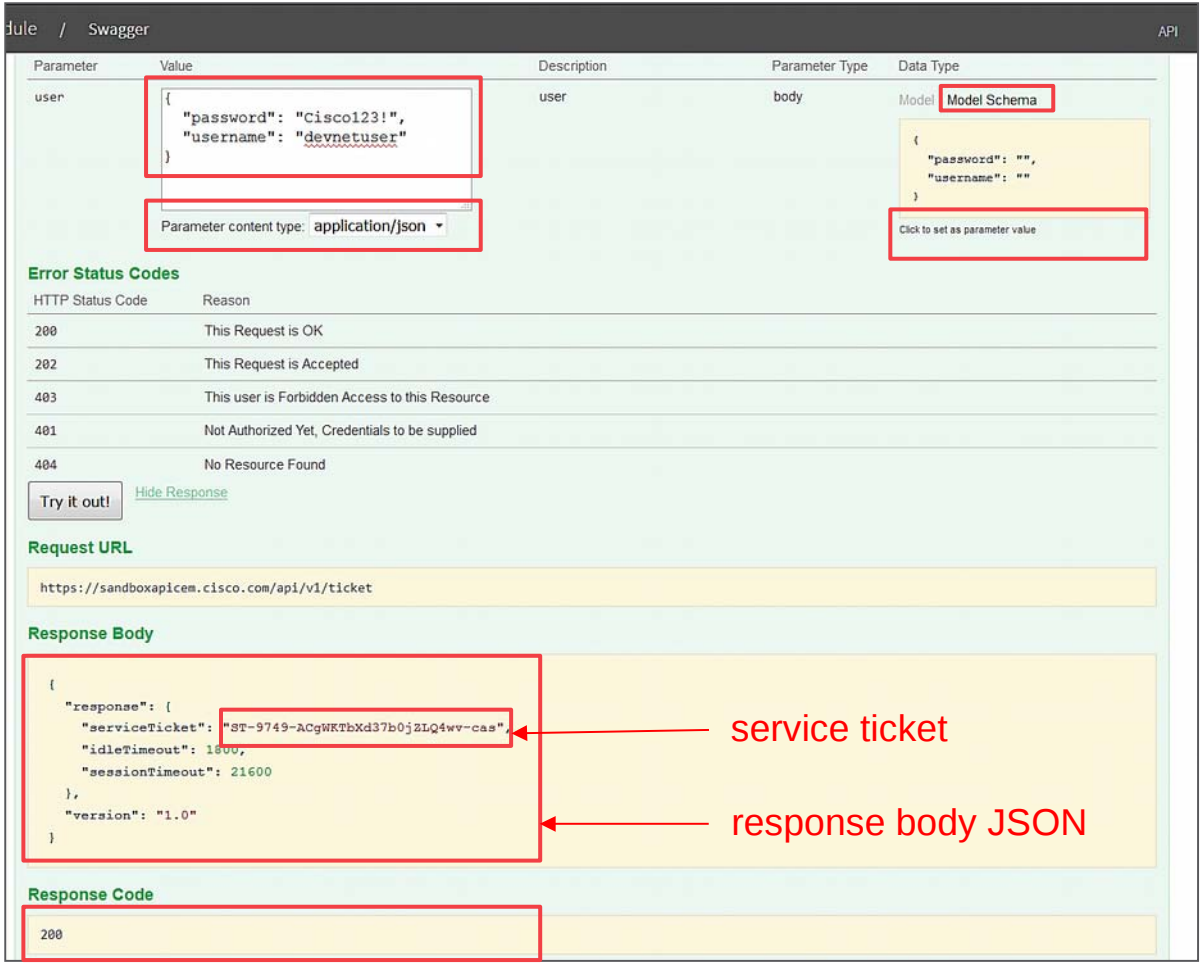
Client application delegates authentication to authentication provider (twitter/google/ facebook/spark/...)

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.



2.0 Programming the APIC-EM REST API

2.4 LAB1 : Getting a Service Ticket with Python

2.4.1 Use POSTMAN to get Service Ticket

2.4.2 Use Python to get Service Ticket

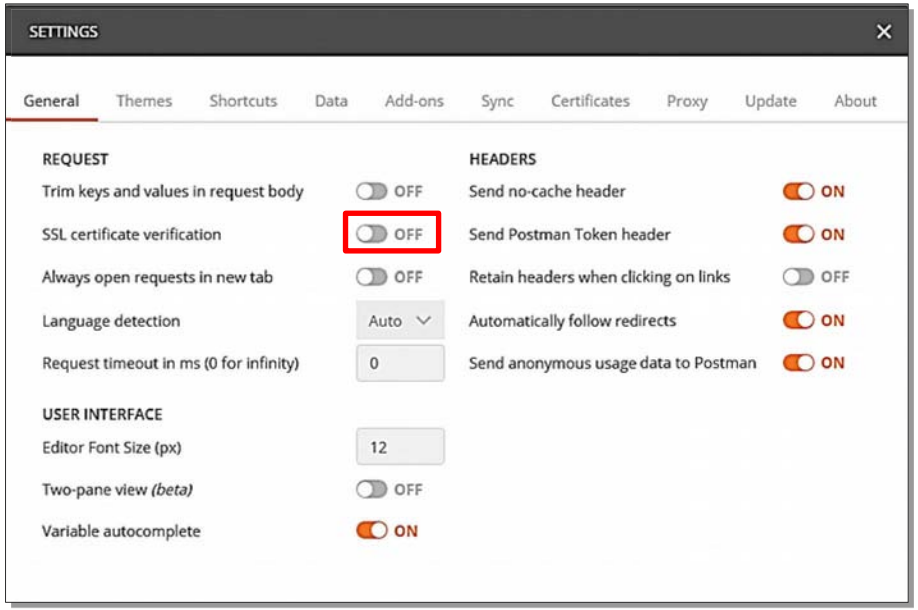
cisco Networking Academy | DEVNET

Postman

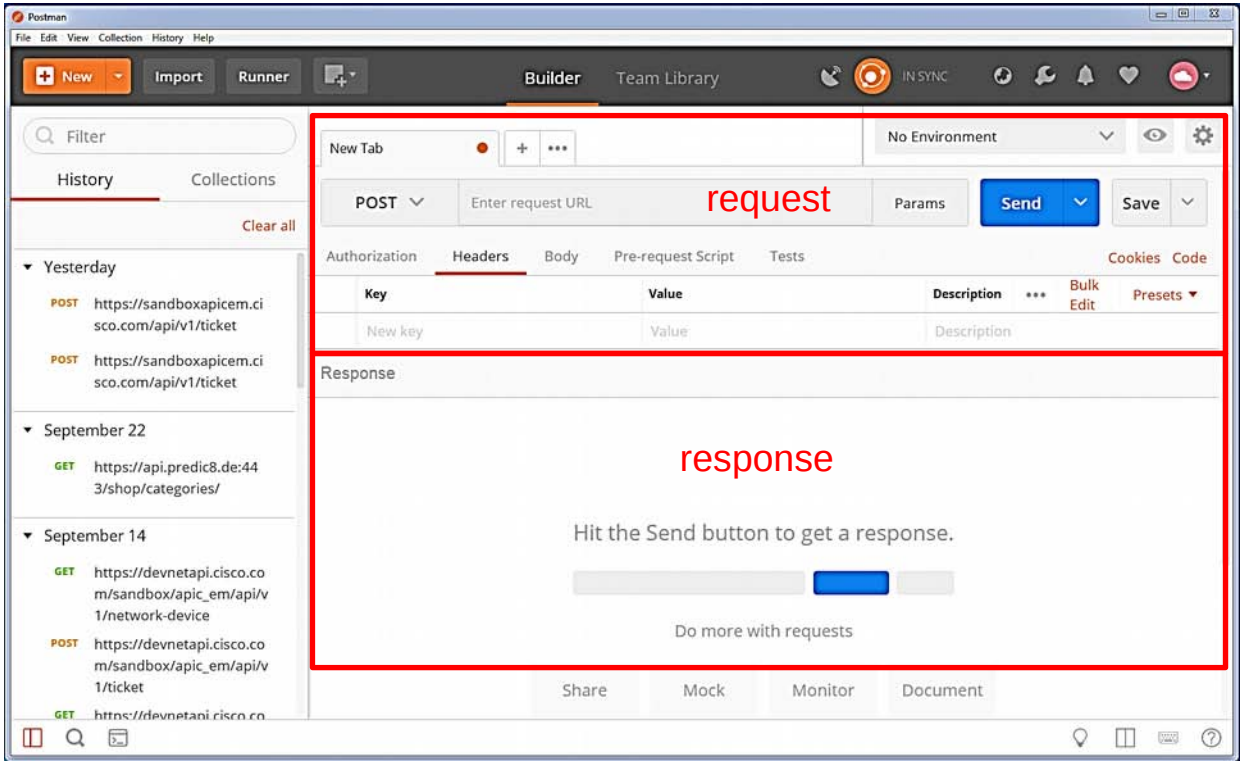
- An HTTP client for MacOS, Linux, Windows that provides an easy way to interact with REST APIs.
- Allows for headers to be easily constructed.
- Displays request status code and response data.
- Frequently used requests can be saved in tabs, history, or collections for reuse.

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"

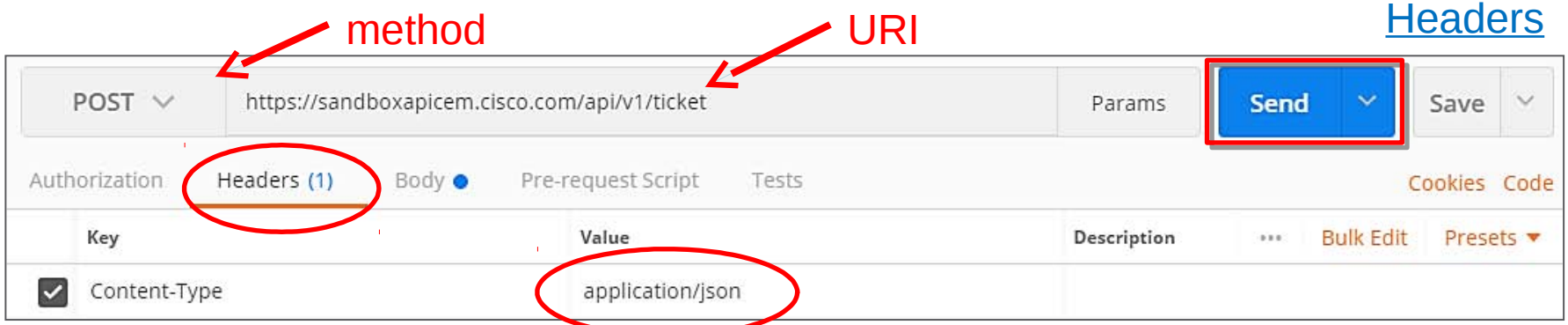


Postman Features

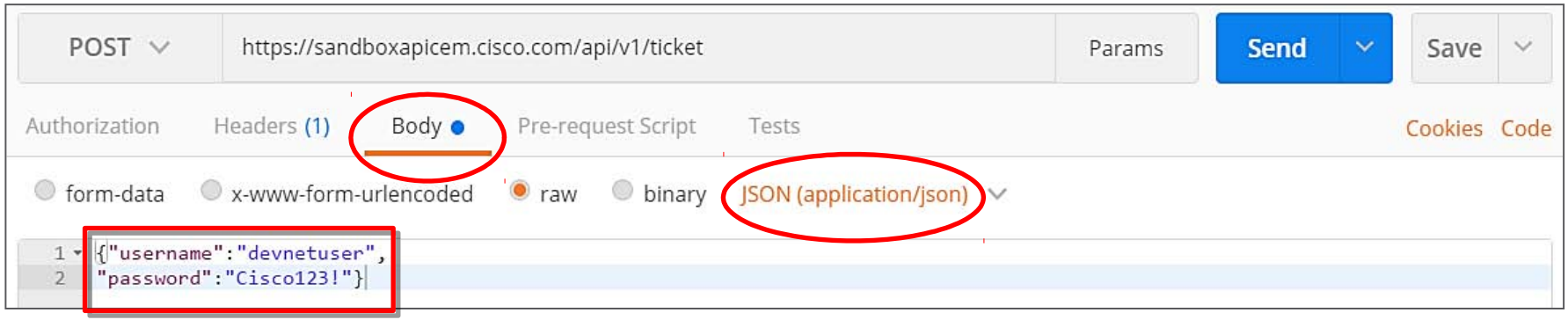


- History
- Tabs
- Collections
- Presets
- Code
- Environments
- Collaboration

Step 2 : Using Postman to get a Service Ticket: Enter Required Information and Send Request

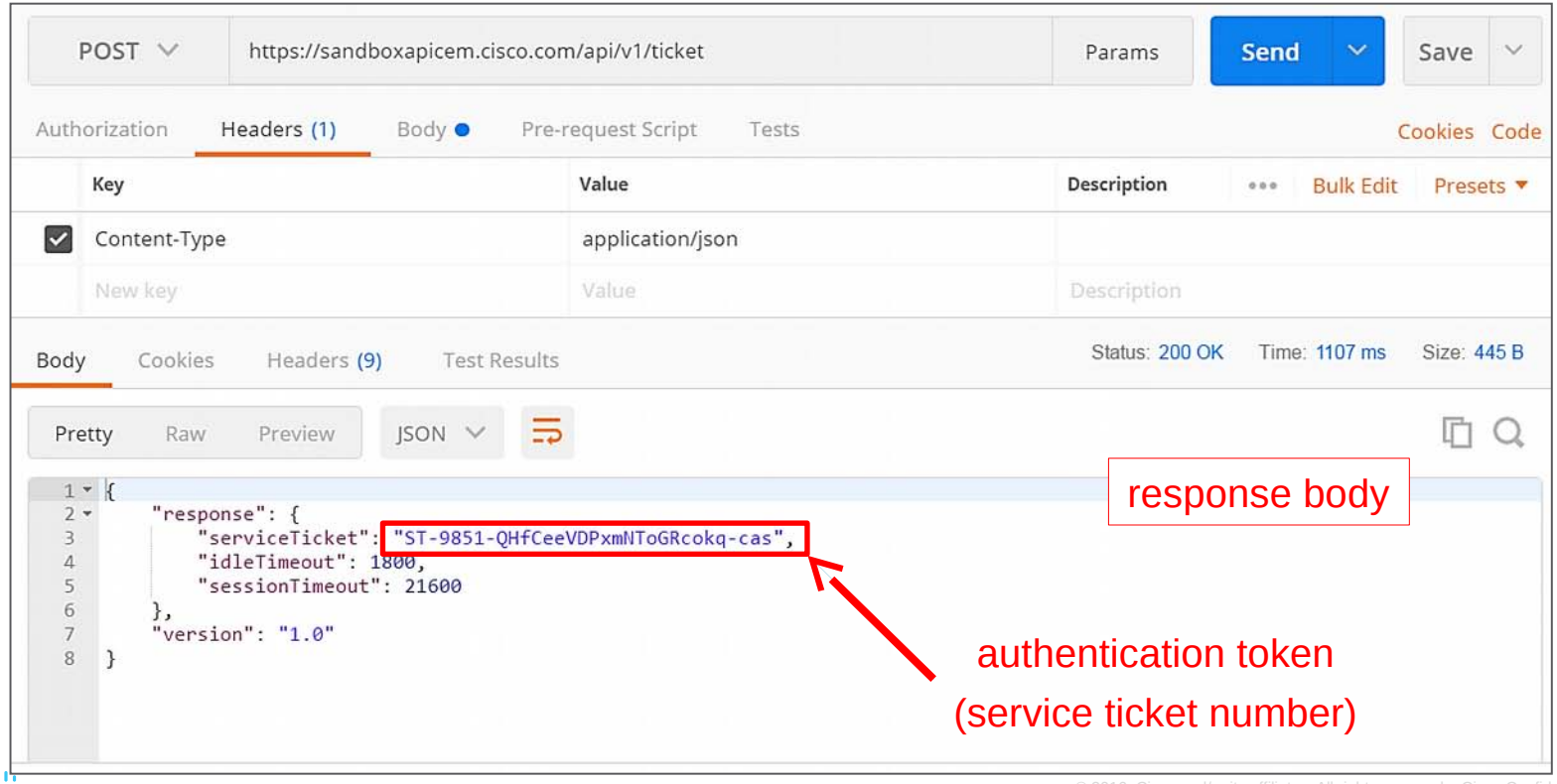


Body JSON: {"username": "devnetuser", "password": "Cisco123!"}



47

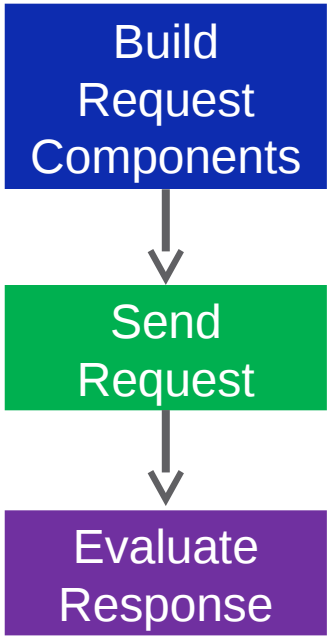
View the Response



48

Overview of the Request Process

1. Build request
   - Method
   - URL
   - Headers
   - Body
   - Authentication
2. Send request
3. Evaluate response
   - Response code
   - Desired data features



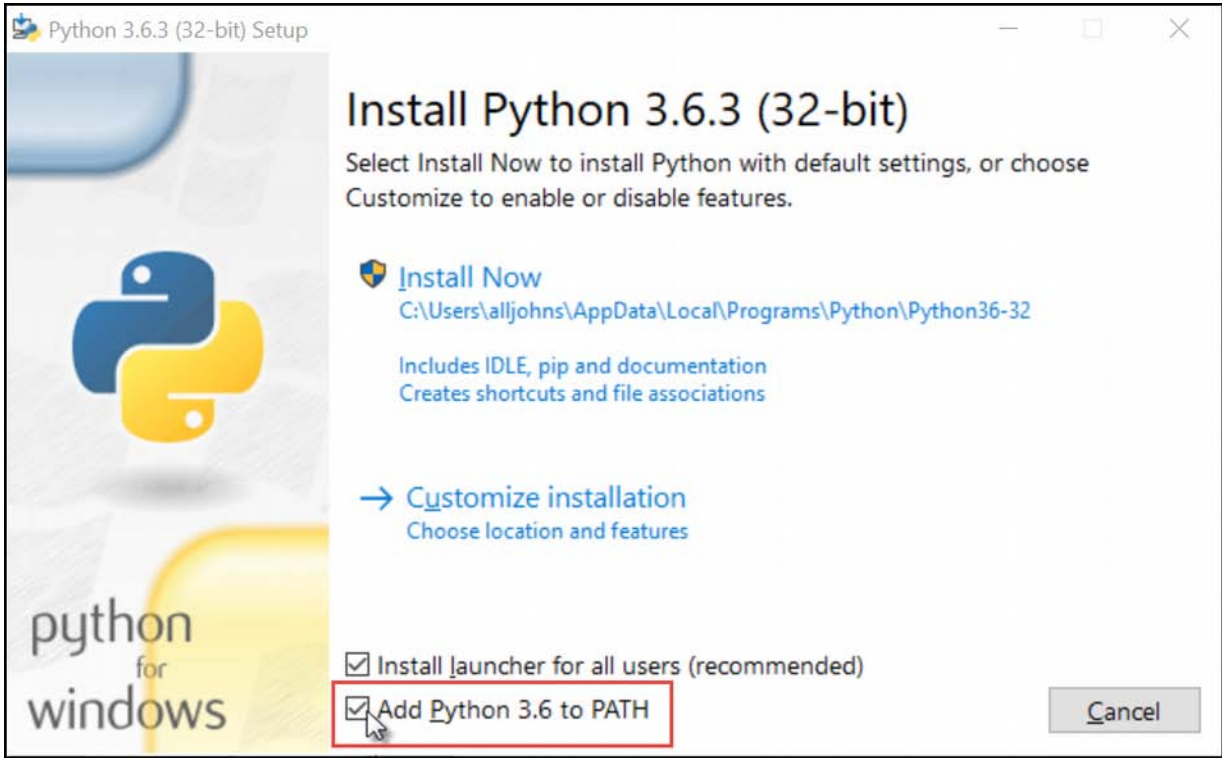
2.0 Programming the APIC-EM REST API

2.4 LAB1 : Getting a Service Ticket with Python

2.4.1 Use POSTMAN to get Service Ticket

2.4.2 Use Python to get Service Ticket

Python IDLE



Python dictionary

```
ipAddress = {"R1":"10.1.1.1","R2":"10.2.2.1","R3":"10.3.3.1"}

print(ipAddress["R1"])
        10.1.1.1

ipAddress["R1"]="10.0.0.1"
```

Python for loop

```
devices=["R1","R2","R3","S1","S2"]
for item in devices:
    print(item)
```

Lab 1: Getting a Service Ticket with Python

```
"""
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: Build the Request Components

```
post_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: Send the Request

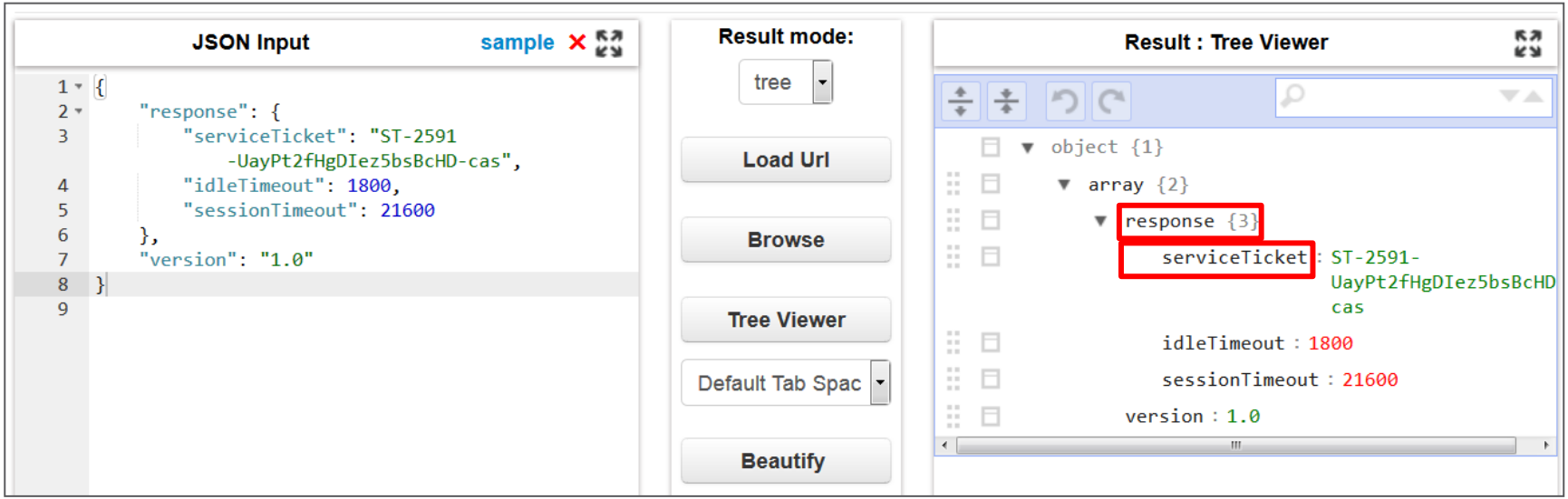
```
resp = requests.post(post_url, json.dumps(body_json),headers=headers,verify=False)
```

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.

About the JSON

https://codebeautify.org/jsonviewer




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

Lab 1: Getting a Service Ticket with Python: Evaluate the Response

```
status = resp.status_code
print ("Ticket request status: " + status\n)

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: 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?

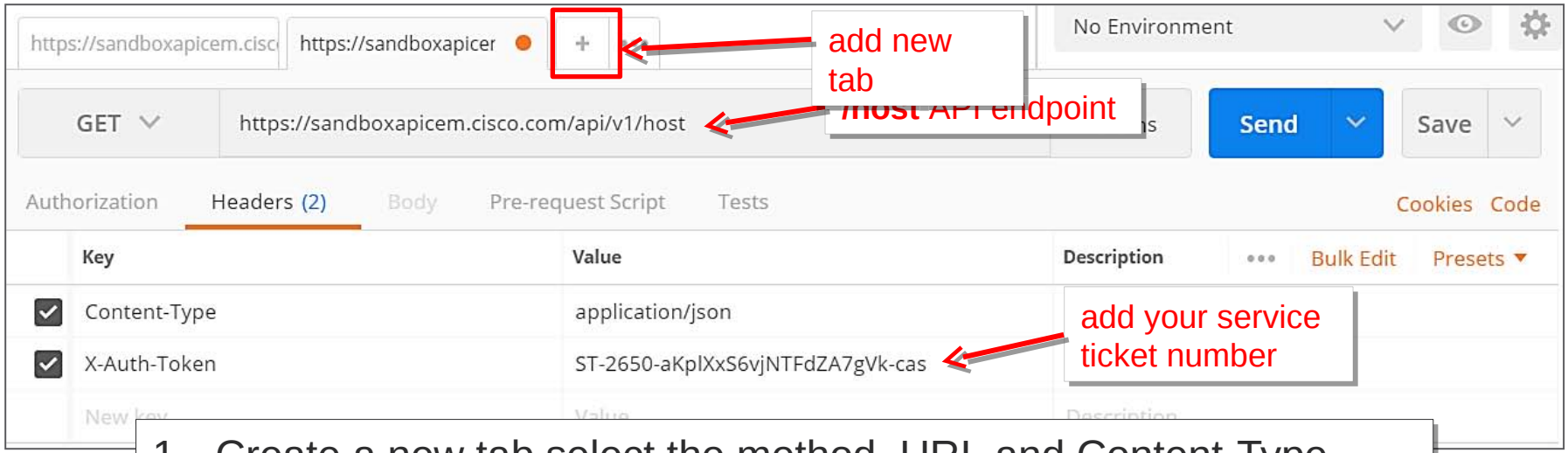
- We will create a small program that requests and displays a table of hosts on the network. We convert this to a function and add it to our functions file.
- We will reuse code to create a small program that requests and displays a table of network devices on the network. We convert this to a function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

2.0 Programming the APIC-EM REST API

2.5 Lab 2 : Create a host inventory in Python

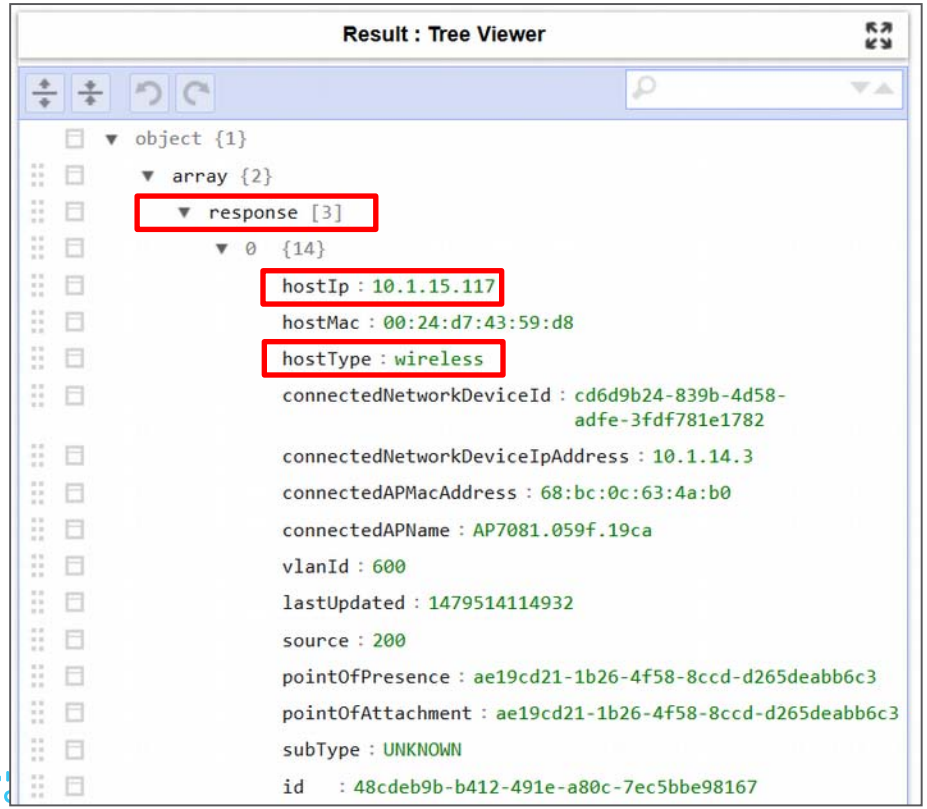
- 2.5.1 Use POSTMAN to get host inventory
- 2.5.2 Use Python to get host inventory

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.

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']
```

2.0 Programming the APIC-EM REST API

2.5 Lab 2 : Create a host inventory in Python

2.5.1 Use POSTMAN to get host inventory

2.5.2 Use Python to get host inventory

Lab 2 : Create Host Inventory in Python

```
'''
02_get_host.py
gets an inventory of hosts from \host endpoint
November, 2017
'''
import requests
import json
import sys
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.

Lab 2 : Create Host Inventory in Python

Build Request Components

```
post_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 2 : Create Host Inventory in Python
Make the Request and Handle Errors

```
try:
    resp = requests.get(post_url,headers=headers,params='',verify = False)
    response_json = resp.json() # Get the json-encoded content from response
    print ('Status of /host request: ',str(resp.status_code))
except:
    print ('Something is wrong with GET /host request!')
    sys.exit()
```

1. Request is made with **get()** method of the **requests** module.
2. A **try:** except: structure is used to handle errors. If an exception is encountered in the **try:** code, the **except:** code executes.
3. Messages are displayed for the status of the request.

Lab 2 : Create Host Inventory in Python Evaluate the Response

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

print (tabulate(host_list,headers=['Number','Type','IP'],tablefmt="rst"))
```

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 2 : Create Host Inventory in Python
Create the Function

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

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

    try:
        resp = requests.get(post_url,headers=headers,params="",verify = False)
        response_json = resp.json()
        print ("Status of /host request: ",str(resp.status_code))
    except:
        print ("Something is wrong with GET /host request!")
        sys.exit()

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

    print (tabulate(host_list,headers=['number','type','host IP'],tablefmt='rst'))
```

1. Copy your program into the functions file.
2. define the function as **get_host()**
3. Indent everything by four *additional* spaces
4. Save the functions file.

2.0 Programming the APIC-EM REST API

2.6 Lab 3 : Create a network-device inventory in Python

Lab 3: Create a Network Device Inventory in Python
Replicate your work for the /**network-device** Inventory endpoint.

1. Save your **get_host.py** file as **get_device.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 **"hostIpAdress"**.
5. Inspect the code and make the substitutions everywhere they are required.
6. Save the file and test. Add the function **get_device()** to your functions file.

2.0 Programming the APIC-EM REST API

2.6 Lab 4 : Path Trace Application

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!

Lab 4: Coding the Path Trace Application: 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 4: Coding the Path Trace Application: 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 4: Coding the Path Trace Application: 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 4: Path Trace Code: Section 1: Setup the Environment

```
#=============================================
# Section 1. Setup the environment and variables required
to interact with the APIC-EM
#=============================================
#++++++++++++Add Values+++++++++++++++
#import modules
                                    
#disable SSL certificate warnings
                                    
#++++++++++++++++++++++++++++++++++++

#++++++++++++Add Values+++++++++++++++
# Path Trace API URL for flow_analysis endpoint
post_url =              #URL of API endpoint
# Get service ticket number using imported function
ticket =                # Add function to get service ticket
# Create headers for requests to the API
headers =               # Create dictionary containing headers for
the request
#++++++++++++++++++++++++++++++++++++
```

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

Lab 4: Path Trace
Code Section 2: Display list of hosts and devices

```
#============================
# Section 2. Display list of devices and IPs by calling
get_host() and get_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 **get_host()** and **get_devices()** functions here.

Lab 4: Path Trace
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 4: Path Trace
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
path = ____________#Convert the path_data to JSON using json.dumps()
resp = ____________#Make the request. Construct the POST request to the API

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

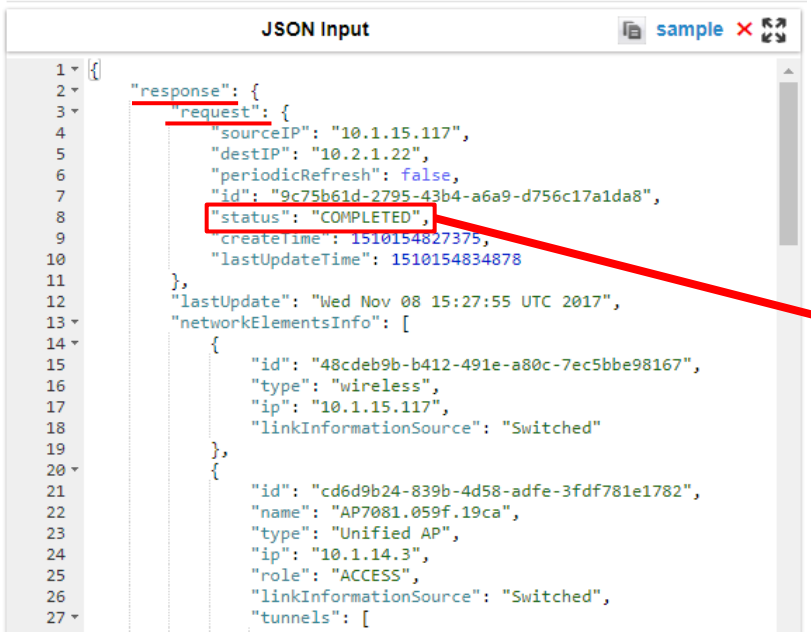
```
#============================
# Section 5. Check status of Path Trace request, output results when
COMPLETED
#============================

#initialize variable to hold the status of the path trace
status = ""

#++++++++++++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 URL that was created in Section 1
#++++++++++++++++++++++++++++++++++++
```

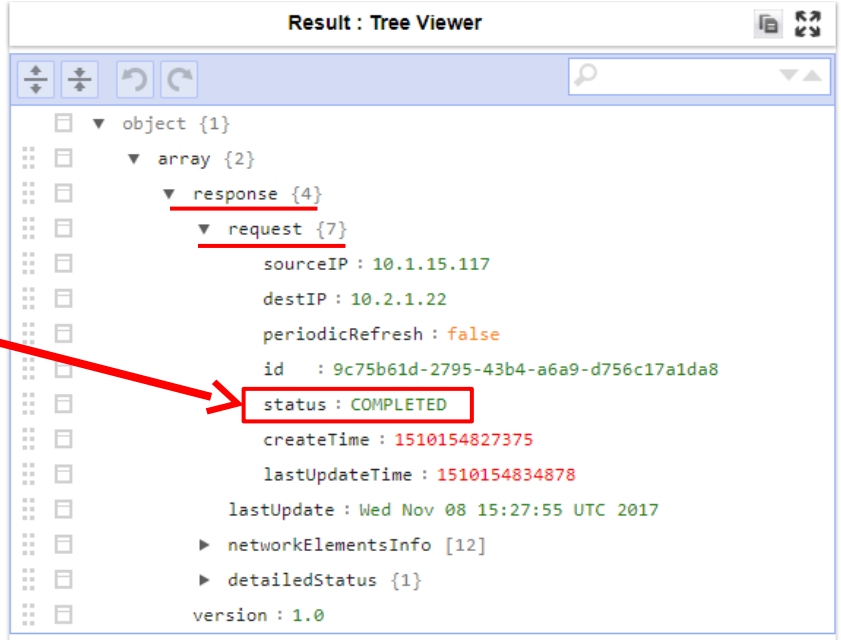
Lab 4: Path Trace
Code Section 5: Check status of Path Trace request - 2

```
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,params="",verify = False)
      response_json = r.json()
      #++++++++++++Add Values+++++++++++++++
      status = [redacted]# 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
            print('Number of status checks exceeds limit. Possible problem with Path Trace.')
            #break
            sys.exit()
      elif status == 'FAILED':
            print('Problem with Path Trace')
            #break
            sys.exit()
      print('REQUEST STATUS: ' + status) #Print the status as the loop runs
```

Lab 4: Path Trace
Code Section 5: Check status of Path Trace request - 3
JSON Status Key

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





Lab 4: Path Trace
Code Section 6: Display Results

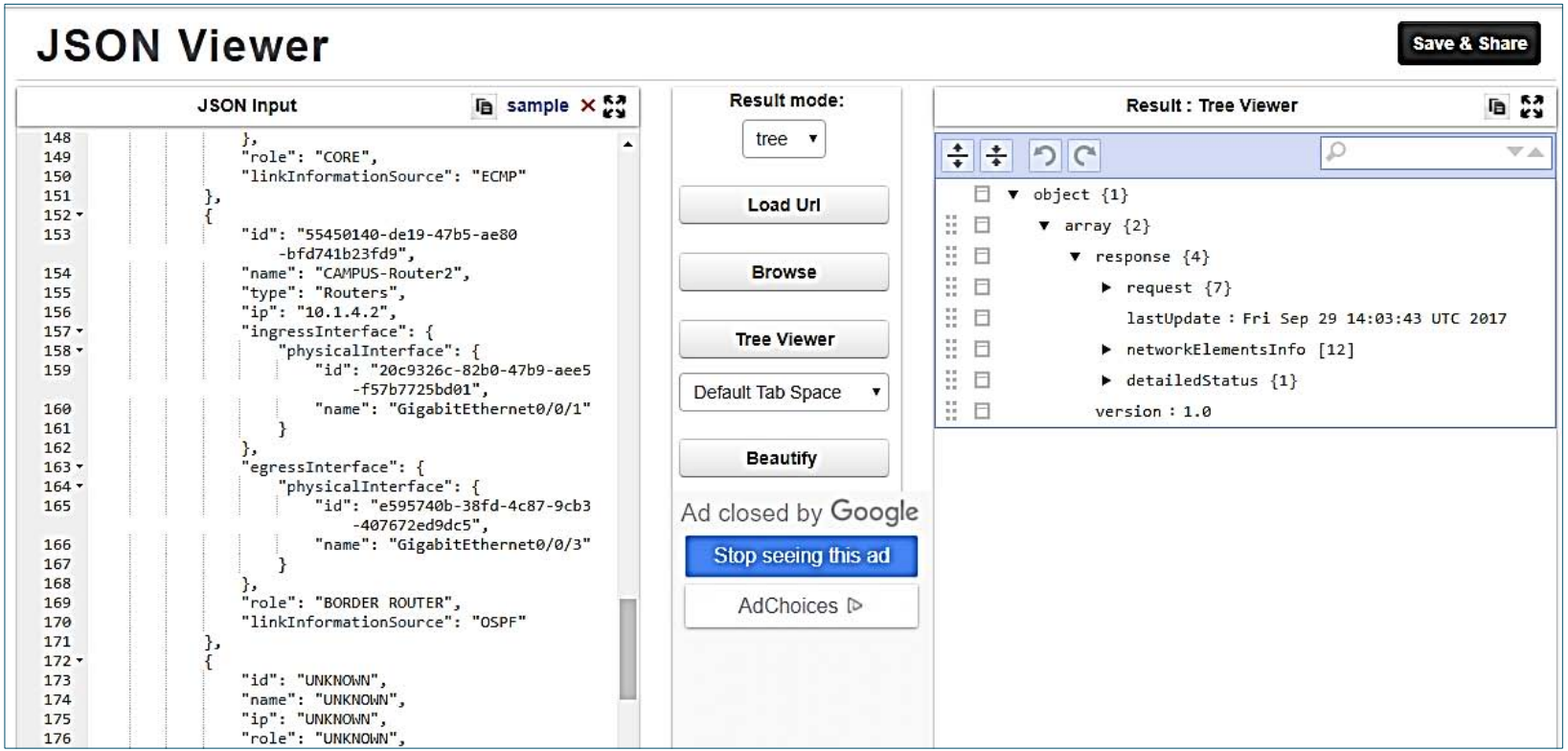
```
#============================
# Section 6. Display results
#============================

# 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 4: Path Trace
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 Path Trace: JSON Practice - Tree View



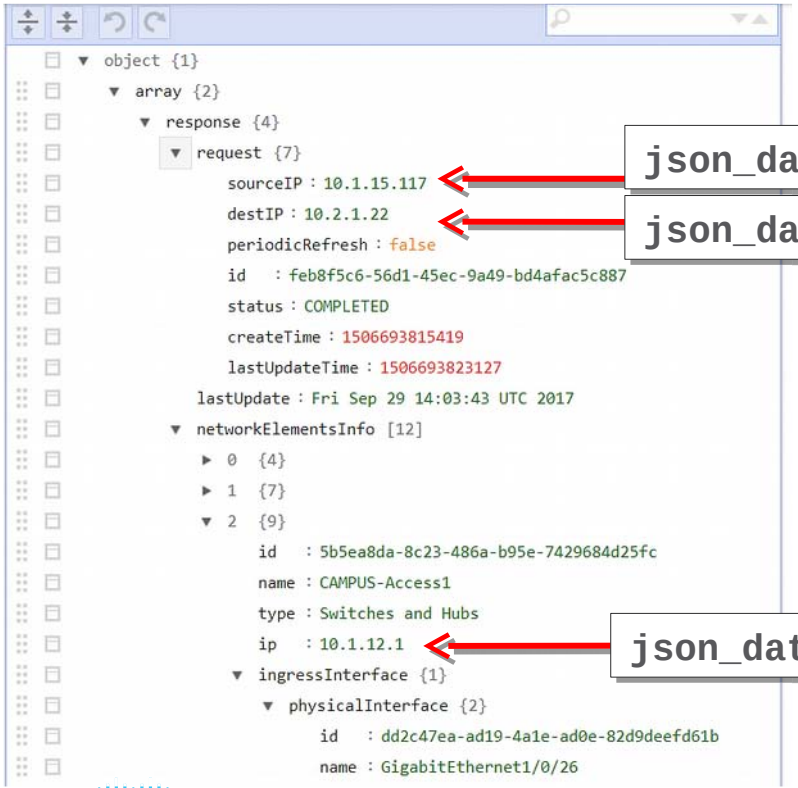
Lab 4: Path Trace
JSON Practice - Load Variables

```
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 4: JSON Practice - Accessing Data in the Response




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

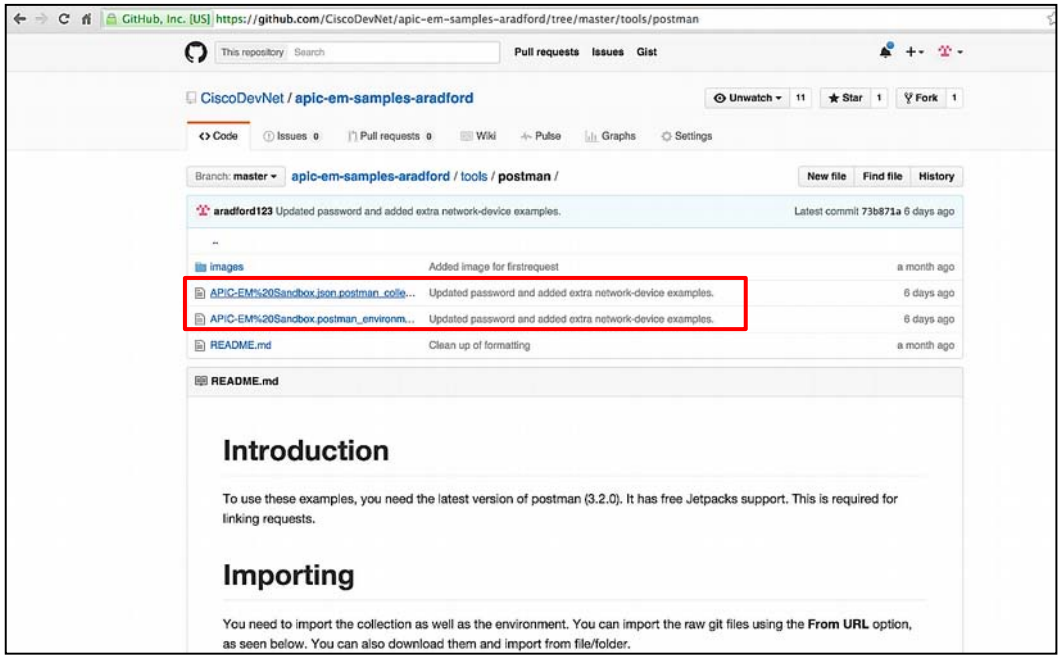
```
json_data['response']['request']['destIP']
```

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']
```

APIC-EM Tools 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

Thank you for attending the workshop!