



>>>network.toCode()

# Network Automation with Python

## Module 3: Working With Network Device APIs



## Module Overview

- HTTP-Based APIs
- non-RESTful HTTP-Based APIs
  - Cisco Nexus NX-API
  - Arista eAPI
- RESTful HTTP-Based APIs
  - Cisco IOS-XE RESTCONF
  - Using Postman
- Consuming HTTP-Based APIs with Python requests



## Module Table of Contents

- [Topic 18: Understanding APIs for Network Devices](#)
- [Topic 19: non-RESTful HTTP APIs](#)
- [Lab 20](#)
- [Topic 20: non-RESTful API Calls with Postman](#)
- [Topic 21: RESTful HTTP APIs](#)
- [Lab 21](#)
- [Topic 22: Consuming HTTP APIs with Python requests](#)
- [Lab 22](#)



A blue-tinted photograph of a dense city skyline, likely New York City, featuring numerous skyscrapers and buildings. The image serves as the background for the slide.

# >>> Lecture 9: Network APIs

*Topic 18 - Understanding APIs for Network Devices*

*Topic 19 - non-RESTful HTTP APIs*

*Lab 20*



# Topic 18: Understanding APIs for Network Devices

*HTTP APIs*

*HTTP Request Types*

*HTTP Response Codes*

*Data Encoding Formats*

## >>> From CLI to API

The industry is transitioning to an API first model

- CLI is for humans
- APIs are for machine to machine communication
- APIs do not replace CLI
- APIs can have a profound impact on operations
- APIs facilitate operational efficiency

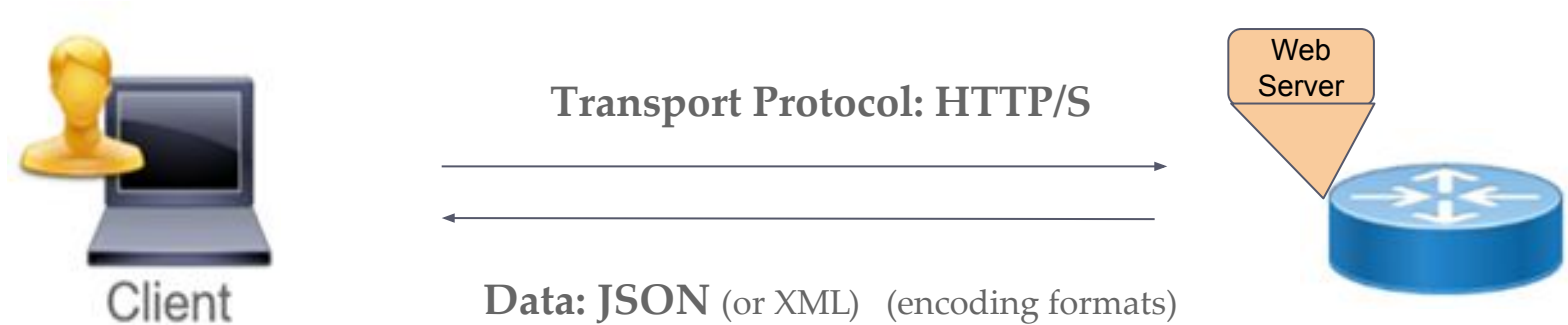
## >>> APIs on Network Devices

- If you understand how to work with a web browser, you understand the concepts of APIs
- Same HTTP Request Methods and Response Codes are used





## >>> Examining an API



*What is the client?*

*What does the data look like?*



## >>> Examining an API



**Python, Ansible, cURL, anything that  
speaks HTTP, e.g. Postman, cURL**

**What is the client?**

### What does the data look like?

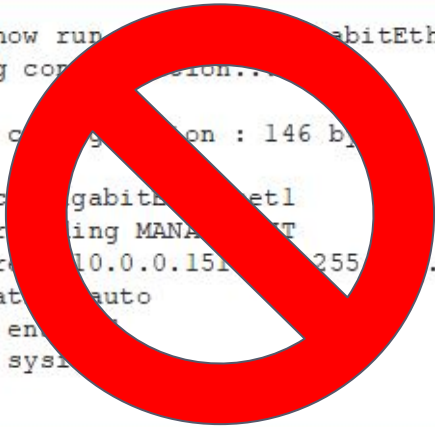
```
cisco#show run interface GigabitEthernet 1
Building configuration...

Current configuration : 146 bytes
!
interface GigabitEthernet1
 vrf forwarding MANAGEMENT
 ip address 10.0.0.151 255.255.255.0
 negotiation auto
 no mop enabled
 no mop sysid
end
```

**This is formatted text, not structured data**

## >>> Examining Data

What does the data look like?



```
cisco#show run | section gigabitEthernet 1
Building configuration...

Current configuration : 146 bytes
!
interface GigabitEthernet1
 vrf forwarding MANA-TEST
 ip address 10.0.0.151/255.0.0.0
 negotiation auto
 no mop enabled
 no mop system
end
```

*Although that can still be  
returned by APIs  
sometimes (and SSH)*

**This is formatted text, not structured data**

## >>> Structured Data: JSON & XML

```
{
  "Cisco-IOS-XE-native:GigabitEthernet": {
    "name": "l",
    "vrf": {
      "forwarding": "MANAGEMENT"
    },
    "ip": {
      "address": {
        "primary": {
          "address": "10.0.0.151",
          "mask": "255.255.255.0"
        }
      }
    },
    "mop": {
      "enabled": false,
      "sysid": false
    }
  }
}
```

JSON

```
<GigabitEthernet>
  <name>l</name>
  <vrf>
    <forwarding>MANAGEMENT</forwarding>
  </vrf>
  <ip>
    <address>
      <primary>
        <address>10.0.0.151</address>
        <mask>255.255.255.0</mask>
      </primary>
    </address>
  </ip>
  <mop>
    <enabled>>false</enabled>
    <sysid>>false</sysid>
  </mop>
</GigabitEthernet>
```

XML



## >>> HTTP-Based APIs

There are two main types of HTTP-Based APIs:

- RESTful HTTP-Based APIs
- non-RESTful HTTP-Based APIs

In other words, those that adhere to the principles of REST and those that do not.

Both use HTTP(s) as transport.

# >>> Sample HTTP Requests

- Authentication Type
- HTTP Request Type
- URL
- Headers
  - Accept
  - Content-Type
- Data (Body)

## Example 1:

```
Basic Auth: ntc/ntc123
Request: GET
Accept: application/json
URL: http://csr1/interfaces/Loopback/100/
```

## Example 2:

```
Basic Auth: ntc/ntc123
Request: POST
Content-Type: application/json
URL: http://device/path/to/resource
Body: {"interface": "Eth1", "admin_state": "down"}
```

## Take note of the body

What **operation** will be performed on a resource?

### Method

### Operation

### Network Example

GET

Retrieve a resource

show interfaces

POST

Create a resource

create interface

PATCH

Update a resource

change interface IP address

PUT

Replace a resource

update full interface configuration

DELETE

Delete a resource

delete an interface

## >>> HTTP Response Codes

Response Code	Description
2xx (200-299)	Success
4xx (400-499)	Client Error
5xx (500-599)	Server Error

Note: the response code types for HTTP-based APIs are no different than standard response codes.



## >>> Data Encoding

Data is sent over the wire as XML or JSON

- JavaScript Object Notation (JSON).
- Open Standard for data communication.
- Uses **name:value** pairs.
- Maps directly to Python dictionaries.

# >>> Topic 19: non-RESTful HTTP APIs

*What makes APIs non RESTful*

*Enable NX-API and use the NX-API Sandbox*

*Enable Arista eAPI and use the eAPI explorer*

## >>> “Network” Clients

### Protocol

### Client

SSH

SecureCRT, Putty, Terminal

SCP

WinSCP, Cyberduck

RDP

Remote Desktop Connection

HTTP

Chrome, Firefox, IE (browsing)

HTTP

APIs: Postman, cURL, on-box clients, Python, Ansible

# >>> Cisco NX-OS NX-API Sandbox

The screenshot shows the Cisco NX-API Sandbox web interface in a browser. The browser's address bar shows the URL `https://nxos-spine1`. The page header includes the Cisco logo, the title "NX-API Sandbox", and user information: "Username: ntc Role: network-admin Version: 9.3(3) Management ip: 10.0.0.15 Hostname: nxos-spine1". On the right side of the header are links for "NX-API References", "Command Reference", and a "Logout" button.

The main content area is divided into two sections. The left section is a large text input field with the placeholder text "Enter CLI commands here, one command per line." A blue callout box points to this field with the text "Enter Commands". The right section contains four dropdown menus for configuring the API call: "Method" (set to "NX-API-CLI"), "Message format" (set to "json-rpc"), "Command type" (set to "cli"), and "Error Action" (set to "Select"). A blue callout box points to these dropdowns with the text "Configurable options".

At the bottom of the interface is a form for sending the API call. It includes a "POST" label, a text input field containing the endpoint `/ins`, and three buttons: "Send" (orange), "Reset" (blue), and "Output Schema" (teal). A blue callout box points to the "POST" label and the endpoint input field with the text "Note the Method and 'endpoint' for all API calls".



# >>> cURL

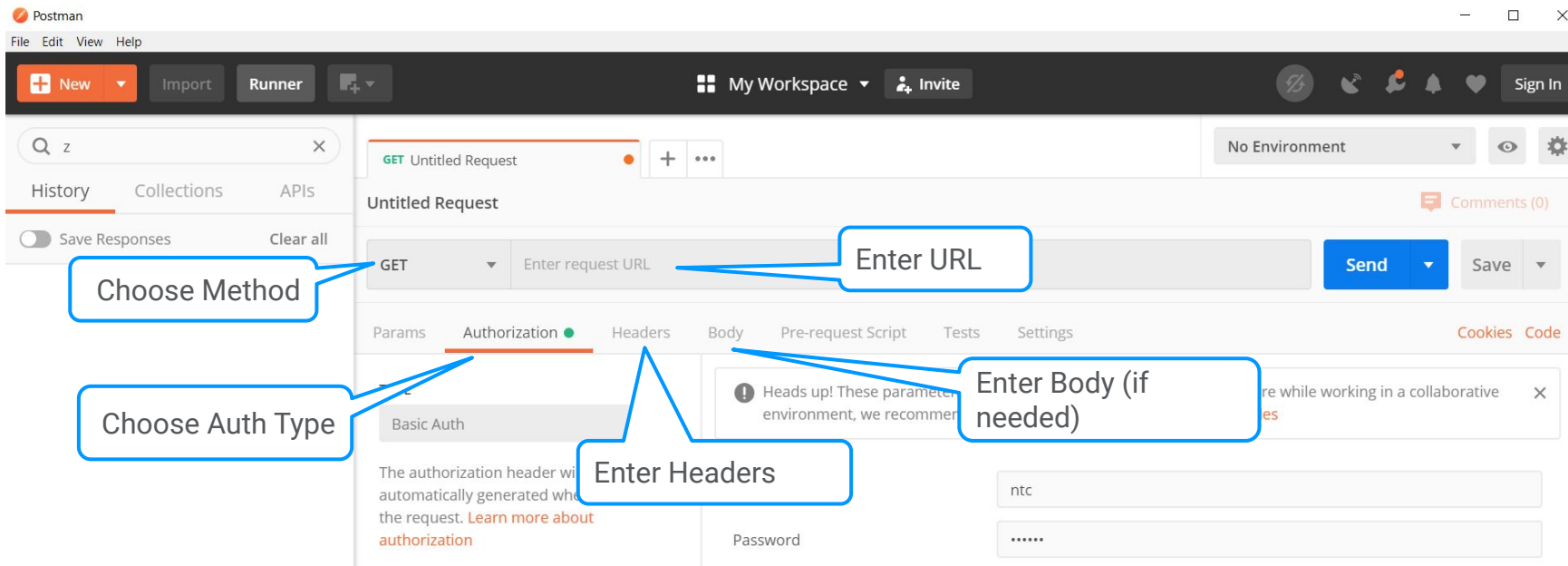
Linux command

Method

```
$ curl -X GET "https://netbox.demo.networktocode.com/api/dcim/sites/" \  
-H "Authorization: Token 123456xyz0123" \  
-H "Accept: application/json; indent=4"
```

Header(s)  
Also uses --header

Use for line breaks and  
readability



## >>> Getting Familiar with JSON Output

- Supported by many vendors who implement web based (REST) APIs
- Certain CLIs allow you to pipe commands to JSON

```
nxos-spine1# show hostname
nxos-spine1.ntc.com
```

```
nxos-spine1# show vlan brief
```

VLAN	Name	Status	Ports
1	default	active	Eth2/5,
	Eth2/6		
100	web_vlan	active	

```
nxos-spine1# show hostname | json
{
  "hostname": "nxos-spine1.ntc.com"
}
```

```
nxos-spine1# show vlan brief | json
{
  "TABLE_vlanbriefxbrief": {
    "ROW_vlanbriefxbrief": [
      {
        "vlandshowbr-vlanid": 16777216,
        "vlandshowbr-vlanid-utf": 1,
        "vlandshowbr-vlanname": "default",
        "vlandshowbr-vlanstate": "active",
      },
      {
        "vlandshowbr-vlanid": 1677721600,
        "vlandshowbr-vlanid-utf": 100,
        "vlandshowbr-vlanname": "web_vlan",
      }
    ]
  }
}
output modified for brevity
```

## >>> Cisco NX-API Developer Sandbox

- On-box web utility that allows you to practice making API calls
- Visually see response objects before writing code
- Simply browse to the Nexus switch using a web browser

# >>> Cisco Nexus NX-API

## Enable NX-API

```
feature nxapi
```

## Configure ports as needed:

```
nxapi https port 8443  
nxapi http port 8080
```

## Certain platforms require a command to enable the sandbox:

```
nxapi sandbox
```

## Certain platforms have VRF support:

```
n9k(config)# nxapi ?  
  certificate Https certificate configuration  
  http        Http configuration  
  https       Https configuration  
  user-vrf    Vrf to be used for nxapi communication
```

# >>> Cisco NX-API Developer Sandbox

The screenshot displays the Cisco NX-API Developer Sandbox interface. At the top, the header includes the Cisco logo, the title "NX-API Sandbox", and user information: "Username: ntc Role: network-admin Version: 9.3(3) Management ip: 10.0.0.15 Hostname: nxos-spine1". Navigation links for "NX-API References", "Command Reference", and a "Logout" button are also present.

The main interface is divided into several sections:

- Command Input:** A text area contains the command "show version". A red box labeled "2" highlights this input, with an arrow pointing to a second red box labeled "3" that highlights the "Send" button.
- Configuration:** On the right, there are dropdown menus for "Method" (set to "NX-API-CLI"), "Message format" (set to "json-rpc"), "Command type" (set to "cli"), and "Error Action" (set to "Select").
- Request:** Below the input area, a "POST" method is selected, and the endpoint "/ins" is entered. The "Send" button is highlighted with a red box and a red circle labeled "3".
- Response:** The "Response" section shows the JSON-RPC output. A red box highlights the "Response:" label. The response body includes details about the Cisco Nexus Operating System (NX-OS) Software, including support links, BIOS version, kickstart version, and file names.

At the bottom, a copyright notice reads: "Copyright © 2014-2018 Cisco Systems, Inc. All rights reserved."

## >>> Arista eAPI Command Explorer

- On-box web utility that allows you to practice making API calls
- Visually see response objects before writing code
- Simply browse to the Arista switch using a web browser

- Enable eAPI

```
management api http-commands
  protocol http
  no shutdown
  vrf MANAGEMENT
    no shutdown
!
```



# >>> eAPI Command Explorer

**ARISTA**  
Command API

ExplorerOverviewCommand Documentation

Simple RequestScript Editor

### Simple eAPI request editor

This page lets you craft a single eAPI request, and explore the returned JSON. Note that this form creates real eAPI requests, so any configuration you perform will apply to this switch. Don't know where to start? Read the [API overview](#) or try one of these examples: [Check version](#), [Create an ACL](#), [Show virtual router](#), or [View running-config](#)!

**API Endpoint**

**Version**

**Commands**  
1 show version

**Format**

**Timestamps**

**AutoComplete**

**ExpandAliases**

**ID**

**Submit POST request**

**Request Viewer**

```
1- {
2  "jsonrpc": "2.0",
3  "method": "executeCommand",
4  "params": {
5    "autoComplete": false,
6    "expandAliases": false,
7    "cmds": [
8      "show version"
9    ],
10   "version": 1
11 },
12 "id": "EapiExplorer-1"
13 }
```

**Response Viewer**

```
1- {
2  "jsonrpc": "2.0",
3  "id": "EapiExplorer-1",
4  "result": [
5    {
6      "cEosToolsVersion": "1.1",
7      "uptime": 1687352.71,
8      "modelName": "cEOSLab",
9      "internalVersion": "4.21.0F-9441269.4210F",
10     "memTotal": 8167884,
11     "serialNumber": "N/A",
12     "systemMacAddress": "52:54:00:79:6f:f2",
13     "bootupTimestamp": 1542263420,
14     "memFree": 6212236,
15     "version": "4.21.0F",
16     "architecture": "i386",
17     "isIntlVersion": false,
18     "internalBuildId": "0e81f168-216d-4896-b345-5b70ca08f5df",
19     "hardwareRevision": ""
20   }
21 ]
22 }
```

## >>> Demo

- Cisco Nexus NX-API Sandbox
- Arista eAPI Command Explorer

Note: these are learning and development / testing tools.



# Lecture 9: Network APIs

*Lab 20*

## >>> Lab Time

- Lab 20 - Exploring eAPI and NXAPI
  - Lab 20.1 - Exploring the Arista eAPI
  - Lab 20.2 - Exploring the Cisco Nexus NX-API
- Please complete **both** of these labs.





# >>> Lecture 10: Exploring Postman

*Topic 20 - non-RESTful API Calls with Postman*

*Topic 21 - RESTful HTTP APIs*

*Lab 21*



# Topic 20: non-RESTful API Calls with Postman

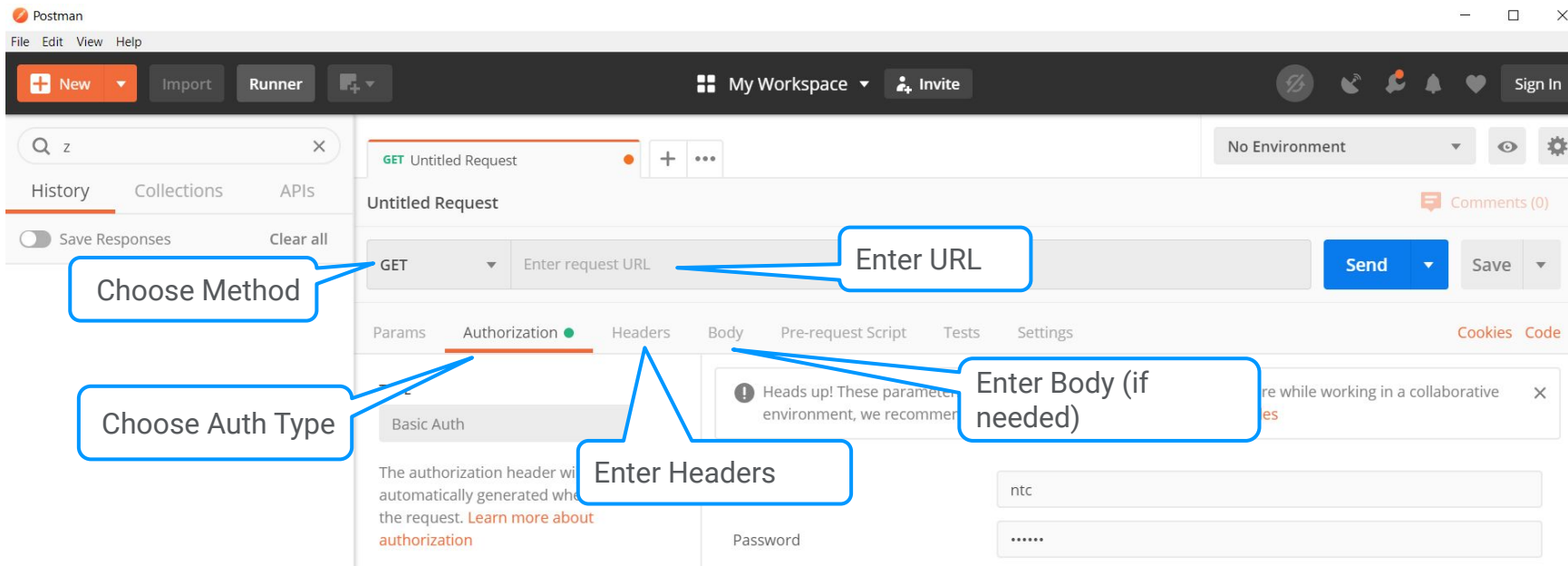
*What is Postman*

*Using the Postman GUI*

## >>> Postman

- User intuitive GUI application to interact with HTTP-based APIs.
- Used for testing, development, and learning.
- You can create a job collection to organize and share with others
- Get it for free - <https://www.postman.com/downloads/>





# >>> Demo

- Prototype API Requests from sandboxes into Postman
  - Cisco Nexus NX-API Sandbox
  - Arista eAPI Command Explorer



# >>> Topic 21: RESTful HTTP APIs

*What is RESTCONF*

*Enable and Use the RESTCONF API on IOS-XE*

## >>> RESTful APIs

- The structure of modern web-based REST APIs came from a PhD paper called [Architectural Styles and the Design of Network-based Software Architectures](#) by Roy Fielding in 2000.
- Goal is to define the detail of working with networked systems on the Internet that use the architecture defined as REST
- REST architecture includes six (6) constraints that must be adhered to. Three (3) of them that help understand REST for this course:
  - Client-Server
  - Stateless
  - Uniform interface

## >>> REST Architecture

- **Client-Server** - Having a client-server architecture allows for **portability and changeability of client applications** without the server components being changed. This could mean having different clients that consume the server resources (back-end API).
- **Stateless** - the communication between the client and server must be stateless. Clients that use stateless forms of communications **must send all data required for the server to understand and perform the requested operation in a single request**. This is in contrast to interfaces such as SSH where there is a persistent connection between a client and a server.
- **Uniform interface** - individual resources in scope within an API call are identified in HTTP request message. For example, in RESTful HTTP-based systems, **the URL used will reference a particular resource**. In the context of networking, the resource maps to a network device construct such as a hostname, interface, routing protocol configuration, or any other resource that exists on the device. The uniform interface also states that the client should have enough information about a resource to create, modify, or delete a resource.

## >>> What is REST?

- **RE**presentational **S**tate **T**ransfer
- Architectural style for information resources
- Perform operations on resources in a stateless manner
- Think:
  - **C**reating a new interface
  - **R**eading information about an interface
  - **U**psdating the description of an interface
  - **D**eleting an interface
- Interact with RESTful services via HTTP

## >>> They are HTTP APIs

- Are you using a POST to “retrieve data”?
- Is it always the same URL?
- Are show commands being sent via HTTP?
- Does it only use GET and POST?

**If so, it's more than likely not RESTful.**

**If different, hierarchical URLs are used to work with different resources and the API supports multiple methods, it's probably RESTful.**



## >>> What is RESTCONF?

- Functional sub-set of NETCONF
- Exposes YANG models via a REST API (URL)
- Uses HTTP(S) as transport
- Uses XML or JSON for encoding
- Uses standard HTTP verbs in REST APIs
- Content-Type & Accept Headers:
  - application/yang-data+json
  - application/yang-data+xml

## >>> RESTCONF on IOS-XE

### Enabling RESTCONF

```
restconf
!  
username <username> privilege 15 password  
<password>  
!  
ip http server  
ip http secure-server  
!
```

## >>> RESTCONF Example 1

Retrieve a full running configuration modeled as JSON.

Method: GET

URL: 'http://csr1/restconf/data/Cisco-IOS-XE-native:native?content=config'

Accept-Type: application/yang-data+json

```
"interface": {  
  "GigabitEthernet": [  
    {  
      "name": "1",  
      "description": "MANAGEMENT_INTEFACE__DO_NOT_CHANGE",  
      "ip": {  
        "address": {  
          "dhcp": {}  
        }  
      },  
      "mop": {  
        "enabled": false,  
        "sysid": false  
      },  
      "Cisco-IOS-XE-cdp:cdp": {  
        "enable": true  
      },  
      "Cisco-IOS-XE-ethernet:negotiation": {  
        "auto": true  
      }  
    },  
    {  
      "name": "10",  
      "shutdown": [  
        null  
      ],  
      "ip": {  
        "no-address": {  
          "address": false,  
          #output removed for example  
        }  
      }  
    }  
  ]  
}
```

## >>> RESTCONF Example 2

The depth-query parameter is used to limit the depth of subtrees returned by the server.

Method: GET

URL:

'http://csr1/restconf/data/Cisco-IOS-XE-native:native?content=config&depth=3'

Accept-Type: application/yang-data+json

- The value of the “depth” parameter is either an integer between 1 and 65535 or the string “unbounded”
- If not present in URI, the default value is: “unbounded”
- Only allowed for GET/HEAD method

```
{
#output removed for example
"interface": {
  "GigabitEthernet": [
    {
      "name": "1",
      "description": "MANAGEMENT_INTEFACE__DO_NOT_CHANGE",
      "ip": {},
      "mop": {},
      "Cisco-IOS-XE-cdp:cdp": {},
      "Cisco-IOS-XE-ethernet:negotiation": {}
    },
    {
      "name": "10",
      "shutdown": [
        null
      ],
      "ip": {},
      "mop": {},
      "Cisco-IOS-XE-ethernet:negotiation": {}
    },
    {
      "name": "11",
      "shutdown": [
        null
      ],
      "ip": {},
      "mop": {},
      "Cisco-IOS-XE-ethernet:negotiation": {}
    }
  ]
}
```

## >>> RESTCONF Example 2

Narrowing the scope and examining the hierarchy

```
{
  "Cisco-IOS-XE-native:GigabitEthernet": {
    "GigabitEthernet": [
      {
        "name": "3",
        "ip": {
          "address": {
            "primary": {
              "address": "10.2.0.151",
              "mask": "255.255.255.0"
            }
          }
        }
      }
    ]
  }
}
```

#output omitted

### Pattern

Cisco-IOS-XE-native:GigabitEthernet (dict) -> GigabitEthernet (list) -> ip  
(dict) -> address (dict) -> primary (dict)

## >>> RESTCONF Example 3

### Request:

```
Method: GET
URL: 'http://csr1/restconf/data/Cisco-IOS-XE-native:native/interface/GigabitEthernet=3/ip'
Accept-Type: application/yang-data+json
```

### Response:

```
{
  "Cisco-IOS-XE-native:ip": {
    "address": {
      "primary": {
        "address": "10.2.0.151",
        "mask": "255.255.255.0"
      }
    }
  }
}
```

## >>> RESTCONF Example 4

### Understanding PUT, PATCH, POST by Updating an Interface

#### Existing Configuration:

```
interface Loopback100
  ip address 222.22.2.2 255.255.255.0 secondary
  ip address 100.2.2.2 255.255.255.0
```

#### BODY Used for POST, PATCH, PUT:

```
{
  "Cisco-IOS-XE-native:Loopback": {
    "name": 100,
    "ip": {
      "address": {
        "primary": {
          "address": "100.2.2.2",
          "mask": "255.255.255.0"
        }
      }
    }
  }
}
```



## >>> RESTCONF Example 4 - The Result

### Request 1:

**POST** http://csr1/restconf/data/Cisco-IOS-XE-native:native/interface/

**Response** 409; Error: Object Already Exists; No change in config

—

### Request 2:

**PATCH** http://csr1/restconf/data/Cisco-IOS-XE-native:native/interface/Loopback

**Response** 204; No change in config

—

### Request 3:

**PUT** http://csr1/restconf/data/Cisco-IOS-XE-native:native/interface/Loopback=100

**Response** 204;

## >>> RESTCONF Example 4 - The Result

### RESULT FOR THE PUT

#### Existing Configuration

```
interface Loopback100  
  ip address 100.2.2.2 255.255.255.0
```

# >>> Static Route Management

Using RESTCONF to manage static route configuration

Starting Configuration:

```
csr1# show run | inc route  
ip route 0.0.0.0 0.0.0.0 10.0.0.2
```

## >>> RESTCONF Example 5 - PATCHing Routes

PATCH <http://csr1/restconf/data/Cisco-IOS-XE-native:native/ip/route>

Body:

```
{
  "Cisco-IOS-XE-native:route": {
    "ip-route-interface-forwarding-list": [
      {
        "prefix": "172.16.0.0",
        "mask": "255.255.0.0",
        "fwd-list": [
          {
            "fwd": "192.168.1.1"
          }
        ]
      },
      {
        "prefix": "10.0.100.0",
        "mask": "255.255.255.0",
        "fwd-list": [
          {
            "fwd": "192.168.1.1"
          }
        ]
      }
    ]
  }
}
```

## >>> RESTCONF Example 5 - PATCHing Routes (cont'd)

Resulting New Configuration:

```
csr1# show run | inc route
ip route 0.0.0.0 0.0.0.0 10.0.0.2
ip route 10.0.100.0 255.255.255.0 192.168.1.1
ip route 172.16.0.0 255.255.0.0 192.168.1.1
```

## >>> RESTCONF Example 6 - PUTing Routes

Starting Configuration:

```
csr1#show run | inc route
ip route 0.0.0.0 0.0.0.0 10.0.0.51
ip route 10.0.100.0 255.255.255.0 192.168.1.1
ip route 172.16.0.0 255.255.0.0 192.168.1.1
```

## >>> RESTCONF Example 6 - PUTing Routes (cont'd)

PUT `http://csr1/restconf/data/Cisco-IOS-XE-native:native/ip/route`

Body:

```
{
  "Cisco-IOS-XE-native:route": {
    "ip-route-interface-forwarding-list": [
      {
        "prefix": "0.0.0.0",
        "mask": "0.0.0.0",
        "fwd-list": [
          {
            "fwd": "10.0.0.2"
          }
        ]
      }
    ]
  }
}
```

## >>> RESTCONF Example 6 - PUTing Routes (cont'd)

Resulting New Configuration:

```
csr1# show run | inc route  
ip route 0.0.0.0 0.0.0.0 10.0.0.2
```



## >>> Summary

- True REST APIs are powerful
- Be careful using PUTs
- With great power comes great responsibility



# Lecture 10: Exploring Postman

Lab 21

# >>> Lab Time

- Lab 21 - Exploring Postman
  - Lab 21.1 Exploring IOS-XE RESTCONF API
  - Lab 21.2 Exploring Arista eAPI



# >>> Lecture 11: APIs with Python

*Topic 22 - Consuming HTTP APIs with Python requests*

*Lab 22*





# Topic 22: Consuming HTTP APIs with Python requests

*Python requests*

*Using requests with eAPI*

*Using requests with IOS-XE*

# >>> Python requests

- Python module to interact with HTTP based APIs (REST)
- Useful functions are post and get
  - Function per HTTP verb, i.e. post is used for POST requests and get is used for GET requests
- Optional, helper method for basic Authentication
- Headers used to dictate data encoding

```
import requests
from requests.auth import HTTPBasicAuth

auth = HTTPBasicAuth('ntc', 'ntc123')

headers = {
    'Content-Type': 'application/json',
    'Accept': 'application/json'
}
```

Sample GET:

```
response = requests.get('http://<device>', headers=headers, auth=auth)
```

## >>> Python requests

- data must be a JSON string - must use `json.dumps()`
- data, headers, and auth are defined parameters that must be used within the requests library
- payload is an arbitrary variable that maps back to device API requirements

```
import requests
import json
from requests.auth import HTTPBasicAuth

auth = HTTPBasicAuth('ntc', 'ntc123')

headers = {
    'Content-Type': 'application/json',
    'Accept': 'application/json'
}
payload = {# some dictionary #}

url = 'http://eos-spine1/command-api'

response = requests.post(url, data=json.dumps(payload), headers=headers, auth=auth)
```

## >>> Python requests - Example on Arista eAPI

```
#!/usr/bin/env python
import requests
import json
from requests.auth import HTTPBasicAuth

if __name__ == "__main__":
    auth = HTTPBasicAuth('ntc', 'ntc123')
    headers = {
        'Content-Type': 'application/json'
    }
    payload = {
        "jsonrpc": "2.0",
        "method": "runCmds",
        "params": {
            "version": 1,
            "cmds": [
                "show version"
            ],
            "format": "json",
            "timestamps": False
        },
        "id": "ntc"
    }
    url = 'http://eos-spine1/command-api'
    response = requests.post(url, data=json.dumps(payload),
headers=headers, auth=auth)
    rx_object = json.loads(response.text)
    print('Status Code: ' + str(response.status_code))
```

- Run show version on a Arista switch.
- Print status\_code, text and OS version.
- The text attribute contains the response of a request as a JSON string.
- The status\_code attribute contains the HTTP response code.

```
Status Code: 200
{
  "jsonrpc": "2.0",
  "result": [
    {
      "memTotal": 3895836,
      "version": "4.15.2F",
      "internalVersion": "4.15.2F-2663444.4152F",
      "serialNumber": "",
      "systemMacAddress": "2c:c2:60:28:54:dd",
      "bootupTimestamp": 1477365548.64,
      "memFree": 1621108,
      "modelName": "vEOS",
      "architecture": "i386",
      "internalBuildId": "0ebbad93-563f-4920-8ecb-731057802b9c",
      "hardwareRevision": ""
    }
  ],
  "id": "ntc"
}
```



## >>> Python requests - Example on Arista eAPI

```
#!/usr/bin/env python
import requests
import json
from requests.auth import HTTPBasicAuth
if __name__ == "__main__":
    auth = HTTPBasicAuth('ntc', 'ntc123')
    headers = {
        'Content-Type': 'application/json'
    }
    payload = {
        "jsonrpc": "2.0",
        "method": "runCmds",
        "params": {
            "version": 1,
            "cmds": [
                "show hostname",
                "show vlan"
            ],
            "format": "json",
            "timestamps": False
        },
        "id": "ntc"
    }
    url = 'http://eos-spine1/command-api'
    response = requests.post(url, data=json.dumps(payload), headers=headers, auth=auth)
    rx_object = json.loads(response.text)
    print('Status Code: ' + str(response.status_code))
    result = rx_object['result']
    print("Hostname: ", json.dumps(result[0], indent=4))
    print("VLANs: ", json.dumps(result[1], indent=4))
```

The cmds request parameter is a list.

- Run show hostname and show vlan at the same time.
- Result is a list and can be used to print individual command output.

```
Status Code: 200
Hostname: {
    "hostname": "eos-spine1",
    "fqdn": "eos-spine1.ntc.com"
}
VLANs: {
    "sourceDetail": "",
    "vlans": {
        "1": {
            "status": "active",
            "interfaces": {},
            "dynamic": false,
            "name": "default"
        }
    }
}
```

## >>> Using requests with IOS-XE

```
#!/usr/bin/env python
import requests
import json
from requests.auth import HTTPBasicAuth

if __name__ == "__main__":
    auth = HTTPBasicAuth('ntc', 'ntc123')
    headers = {
        'Accept-Type': 'application/vnd.yang.data+json',
        'Content-Type': 'application/vnd.yang.data+json'
    }

    url = 'http://csr1/restconf/api/config/native/interface'
    response = requests.get(url, headers=headers, auth=auth)
    print('Status Code: ' + str(response.status_code))
    print("\nInterfaces Object: ", response.text)
```

```
Status Code: 200

{
  "ned:interface": {
    "GigabitEthernet": [
      {
        "name": "1"
      },
      {
        "name": "2"
      },
      {
        "name": "3"
      },
      {
        "name": "4"
      }
    ]
  }
}
```



# Lecture 11: APIs with Python

*Lab 22*

## >>> Lab Time

- Lab 22 - Using Python requests:
  - Lab 22.1 - Using Python requests with Arista eAPI
  - Lab 22.2 - Using Python requests with Cisco NX-API