Cisco live!

January 29 – February 2, 2018 · Barcelona

LTRCRT-2700

Model Driven Network Automation with IOS-XE

Tony Roman, Cisco Content Engineer

Tom Bryan, Cisco Technical Leader

Cisco live!

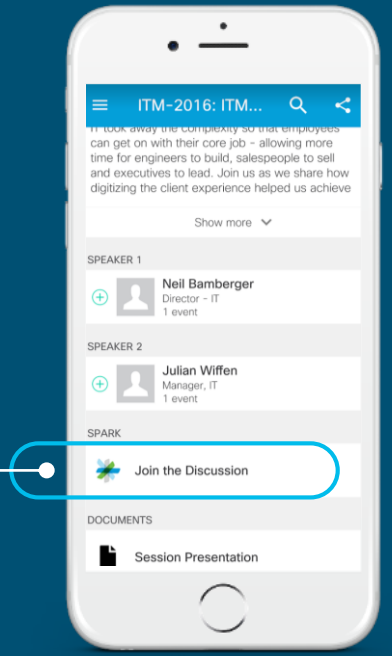
Cisco Spark

Questions?

Use Cisco Spark to communicate with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install Spark or go directly to the space
4. Enter messages/questions in the space



cs.co/ciscolivebot#LTRCRT-2700

Who are we?

Tony Roman

- Learning@Cisco Content Engineer focused on Network Programmability
- DevOps professional turned NetDevOps
- toroman@cisco.com

Tom Bryan

- Cisco Technical Leader, one of the creators of VIRL Personal Edition and Cisco Modeling Lab
- Software developer with a focus on network simulation and tools for engineers
- tombry@cisco.com
- Twitter: @tbryan314

Agenda

- Introduction
  - Data Encoding, Data Models, APIs, and Transport
- Diving Deeper into RESTCONF
- Exploring RESTCONF on IOS-XE with Postman (Lab)
- Automating IOS-XE with RESTCONF while using Python requests (Lab)
- Diving Deeper into NETCONF
- Automating IOS-XE with NETCONF using Python ncclient (Lab)

Introduction

Why are we here?

Network Operations & Management hasn't changed for 20 years.

Unless we count the transition from telnet to SSH.

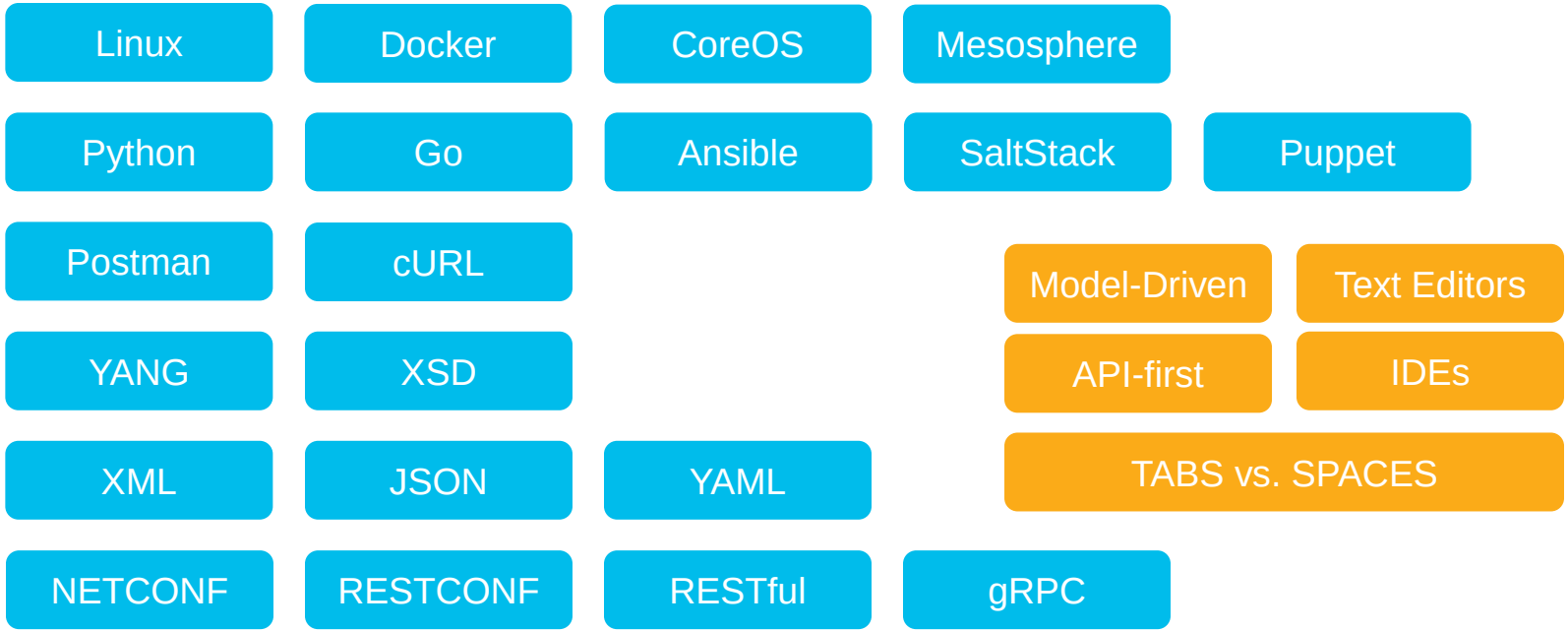
From CLI to API

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

So many new terms…

…in this new world



So many new terms…

…in this new world

Linux

Docker

CoreOS

Mesosphere

Python

Go

Puppet

Postman

cURL

-Driven

Text Editors

YANG

XSD

-first

IDEs

XML

JSON

TABS vs. SPACES

NETCONF

RESTCONF

RESTful

gRPC

Covering a subset of these today

Keep in mind the skills for the future network engineer are increasing

Linux

Python

Postman

YANG

XML

JSON

NETCONF

RESTCONF

RESTful

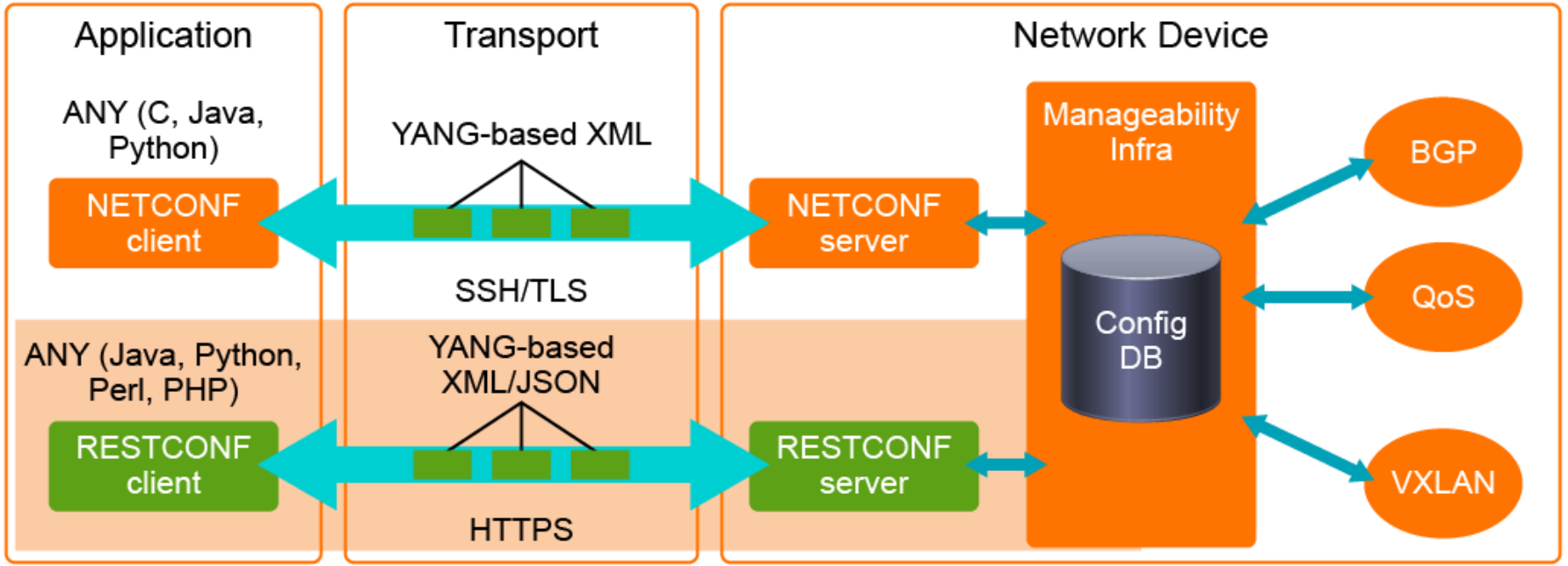
Model-Driven

Text Editors

TABS vs. SPACES

Cisco IOS XE APIs Overview

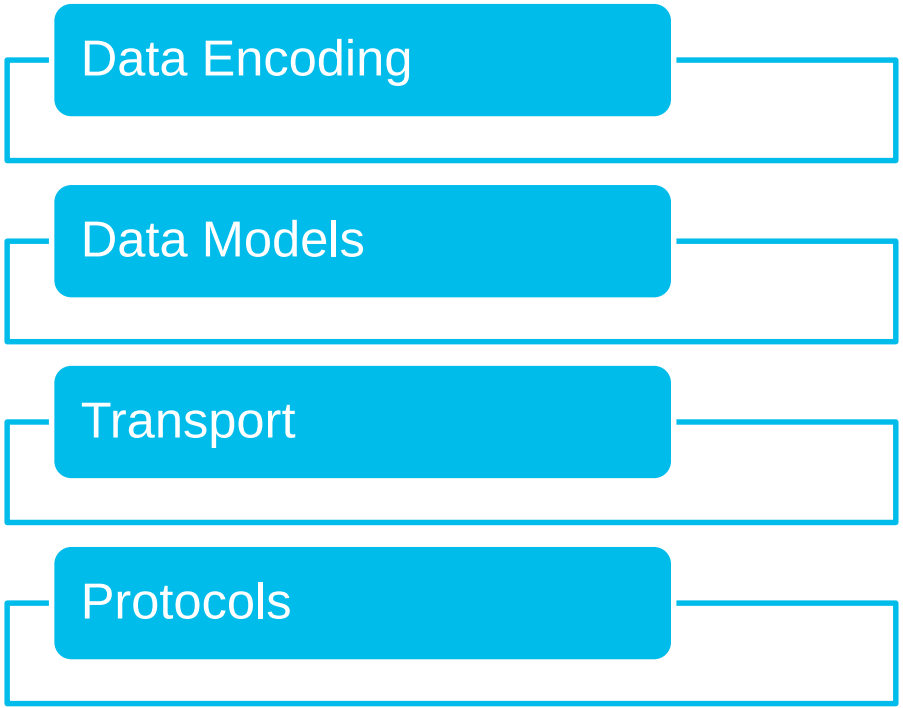
Accept NETCONF and RESTCONF requests and communicates directly to configuration datastore(s).



Objectives

- Understand what data models are and why they're important
- Leave here knowing how to use Postman so you can explore any HTTP-based API
- Become familiar with the basics of reading and executing Python scripts using IOS-XE APIs
- Know enough to continue your IOS-XE API journey tomorrow
- Realize you must understand the basics of programmability sooner than later
- Understand you will not be an IOS-XE API expert in 4 hours ☺
- Recognize learning takes time

Breaking down Network APIs



Data Encoding

Who gets confused seeing “{ }” or “</tag>” in code or text output?

Data Encoding

Who gets confused seeing “`{ }`” or “`</tag>`” in code or text output?

Too bad? 🤓 They are standard and take a little getting used to…

Data Encoding

- There needs to be structure behind what is communicated between **systems**

```
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

Data Encoding: JSON & XML

```
{
  "Cisco-IOS-XE-native:GigabitEthernet": {
    "name": "1",
    "vrf": {
      "forwarding": "MANAGEMENT"
    },
    "ip": {
      "address": {
        "primary": {
          "address": "10.0.0.151",
          "mask": "255.255.255.0"
        }
      }
    },
    "mop": {
      "enabled": false
    },
    "Cisco-IOS-XE-ethernet:negotiation": {
      "auto": true
    }
  }
}
```

JSON

```
<GigabitEthernet>
  <name>1</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>
</mop>
<negotiation>
  <auto>true</auto>
</negotiation>
</GigabitEthernet>
```

XML

Data Encoding: JSON & XML

```
{
  "Cisco-IOS-XE-native:GigabitEthernet": {
    "name": "1",
    "vrf": {
      "forwarding": "MANAGEMENT"
    },
    "ip": {
      "address": {
        "primary": {
          "address": "10.0.0.151",
          "mask": "255.255.255.0"
    "mop": {
      "enabled": false
    }
  }
}
```

JSON

```
<GigabitEthernet>
  <name>1</name>
  <vrf>
    <forwarding>MANAGEMENT</forwarding>
  </vrf>
  <ip>
    <address>
      <primary>
        <address>10.0.0.151</address>
        <mask>255.255.255.0</mask>
      </primary>
    <enabled>false</enabled>
  </mop>
</GigabitEthernet>
```

XML

```
print response['Cisco-IOS-XE-native:GigabitEthernet']['ip']['address']['primary']['address']
10.0.0.151
```

```
print response.findall('.//{http://cisco.com/ns/yang/Cisco-IOS-XE-native}address')[1].text
10.0.0.151
```

Data Encoding

- There needs to be structure behind what is communicated between **systems**

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

Current configuration : 146 bytes

e GigabitEthernet1
arding MANAGEMENT
ss 10.0.0.151 255.255.255.0
ion auto
nabled
ysid
```

```
>>> for line in response.splitlines():
...     if 'address' in line:
...         print line.split('address')[1].split(' ')[1]
...
10.0.0.151
>>>
Other options exist.  All use spacing, loops, or Regular
Expressions.
Still very prone to error.
But, automation was/is still possible.
```

formatted text, not structured data

Data Encoding

*Machines can **easily** parse XML and JSON. You can easily send an object that a machine understands.*

We now know machines communicate using structured data.

What about the proper syntax and constraints of that data?

Data Models

What are **Data Models**?

- Data models ***describe*** a constrained set of data
- Use well-defined parameters to standardize the representation of data

Data Models

Examples

- How do you describe a VLAN?
  - VLAN ID
  - VLAN name
  - VLAN state
  - …

Data Models

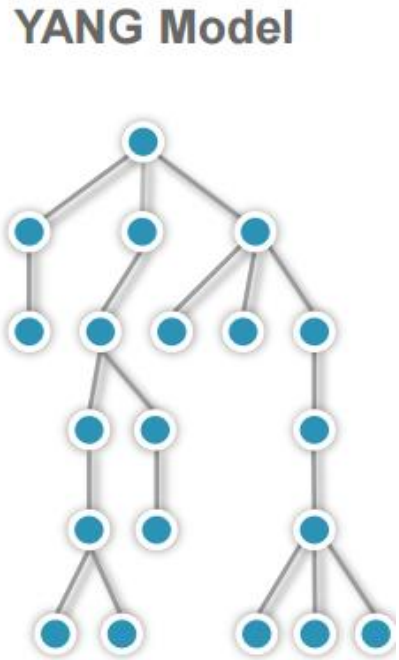
Examples

- How do you describe a VLAN?
  - VLAN ID – integer between 1 and 4096 ?
  - VLAN name – string between 1 and N characters ?
  - VLAN state – enumeration of “down/up” or “shutdown/ no shutdown”
  - …

This is defining the constraints of the data– the **Data Model** for a VLAN.

YANG

- It is a network-centric data modeling language defined in RFC 6020 specifically built for “network data”
- Initially built for NETCONF
  - Now also used by RESTCONF and can be used by any protocol
- Models configuration and operational state data
- Provides syntax and semantics



YANG

Sample YANG Module

```
module: ietf-interfaces
   +--rw interfaces
   |  +--rw interface* [name]
   |     +--rw name                        string
   |     +--rw description?                string
   |     +--rw type                        identityref
   |     +--rw enabled?                    boolean
   |     +--rw link-up-down-trap-enable?   enumeration {if-mib}?
   +--ro interfaces-state
      +--ro interface* [name]
         +--ro name               string
         +--ro type               identityref
         +--ro admin-status       enumeration {if-mib}?
         +--ro oper-status        enumeration
[…]
```

YANG

Open and Native YANG Models

- **Standard definition** (IETF, ITU, OpenConfig, etc.)
- Compliant with standard, i.e. "Policy"

  ietf-diffserv-policy.yang

  ietf-diffserv-classifer.yang

  ietf-diffserv-target.yang

- **Cisco definition**
- Common across Cisco platforms,

  i.e. "OTV" on IOS-XE and NX-OS

- **Cisco definition**
- Unique to specific Cisco platform,

  i.e. "BGP" extensions on IOS-XE

Transport

The properly structured and encoded data still needs a way to get from point A to point B:

- SSH
- HTTP
- HTTPS
- TLS
- HTTP/2

APIs

Two most common types of network APIs

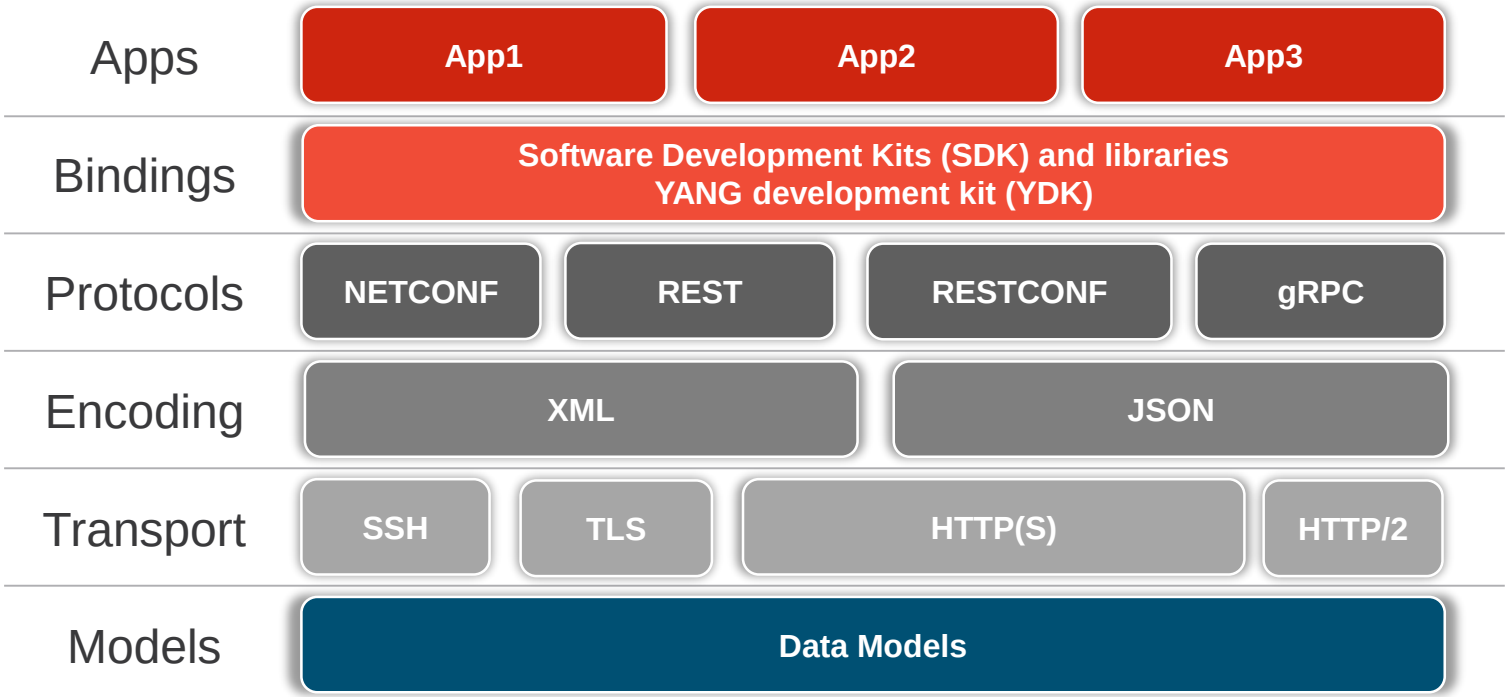
NETCONF

- Uses XML encoded data
- Data that is transported is modeled either using YANG or XSD
- NETCONF on IOS-XE uses XML data that adheres to YANG models
- NETCONF protocol itself has network centric features (cover some of these later)
- Transported over SSH

REST (for network devices)

- Uses XML or JSON encoded data
- Modeled either using YANG, XSD, JSD, or custom modeling language
- RESTCONF on IOS-XE uses XML/JSON data that adheres to YANG models
- IOS-XE produces their REST API using data modeled from YANG → RESTCONF
- Transported over HTTP/HTTPS

Network Programmability Stack



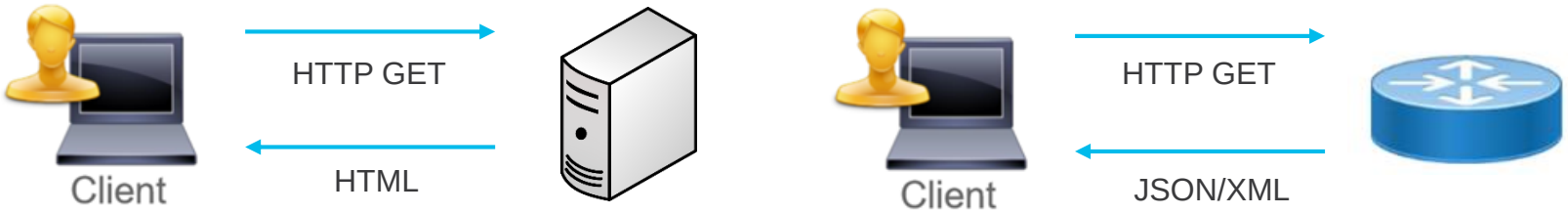
Diving Deeper into RESTCONF

Before RESTCONF…

A little background on REST

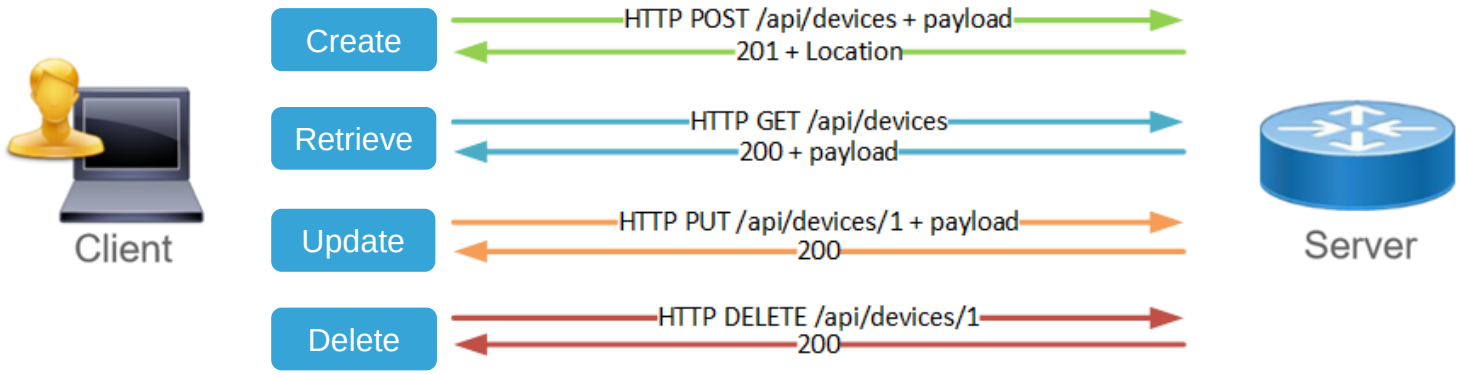
REST

- If you understand how to work with a web browser, you understand REST.
- Same HTTP Request Methods and Response Codes are used



REST

Create, Retrieve, Update and Delete (CRUD)



REST

HTTP Verbs

GET	• Retrieve / Read a resource
POST	• Creates a new resource
PUT	• Update/Replace a resource
PATCH	• Update/Modify a resource
DELETE	• Removes a resource

REST

HTTP Verbs in the context of network devices

GET	• Retrieve / Read a resource	show command
POST	• Creates a new resource	create logical interface
PUT	• Update/Replace a resource	replace full interface config with what's in the body of request
PATCH	• Update/Modify a resource	update (append) interface config with what's in the body of request
DELETE	• Removes a resource	remove logical interface

REST

Common HTTP Response Codes

Success (2xx)	Description
200	Request Succeeded
201	The request has been fulfilled; new resource created
204	The server fulfilled request but does not return a body

Server Error (5xx)	Description
500	Internal Server Error
501	Not implemented

Client Error (4xx)	Description
400	Bad Request. Malformed Syntax
401	Unauthorized
403	Server understood request, but refuses to fulfill it.
404	Resource not found given URI

Constructing an API Request

- URL
- HTTP Verb
  - GET, POST, PATCH, PUT, DELETE
- Body
  - Used when making a configuration change. We'll show a JSON Body soon.
- Headers
  - Define the structure of the data being sent and desired to be received.
  - *Are you using JSON or XML?*
- Authentication
  - Your privileged level 15 credentials

IOS-XE RESTCONF Request 1

Retrieving the GigE1 Configuration

- URL – http://clus1/restconf/api/running/native/interface/GigabitEthernet/1/
- HTTP Verb
  - GET
- Body
  - N/A
- Headers
  - Accept – application/vnd.yang.data+json
- Authentication
  - Your privileged level 15 credentials

IOS-XE RESTCONF Response 1

Retrieving the GigE1 Configuration


```
{
  "Cisco-IOS-XE-native:GigabitEthernet": {
    "name": "1",
    "vrf": {
      "forwarding": "MANAGEMENT"
    },
    "ip": {
      "address": {
        "primary": {
          "address": "10.0.0.151",
          "mask": "255.255.255.0"
        }
      }
    },
    "mop": {
      "enabled": false
    },
    "Cisco-IOS-XE-ethernet:negotiation": {
      "auto": true
    }
  }
}
```

IOS-XE RESTCONF Request 2

Configuring GigE2

- URL – http://clus1/restconf/api/running/native/interface/GigabitEthernet/2/
- HTTP Verb
  - PATCH
- Body
  - →
- Headers
  - Content-Type – application/vnd.yang.data+json
- Authentication
  - Your privileged level 15 credentials

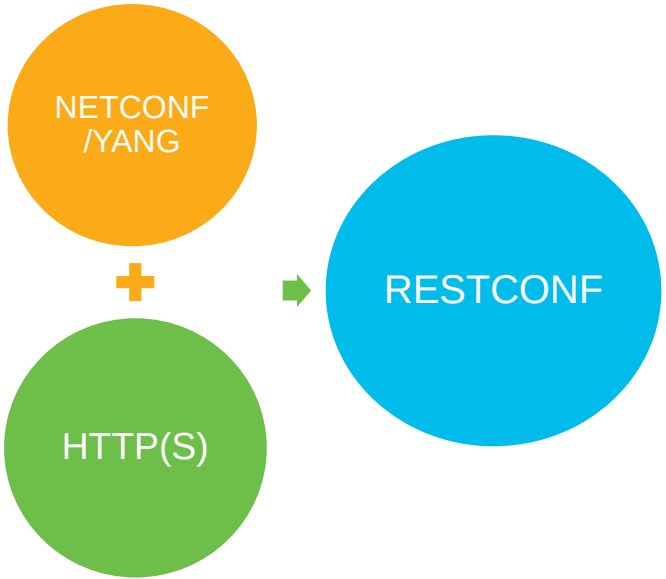
```
{
  "Cisco-IOS-XE-native:GigabitEthernet": {
    "ip": {
      "address": {
        "primary": {
          "address": "10.2.0.151",
          "mask": "255.255.255.0"
        }
      }
    }
  }
}
```

So…REST vs. RESTCONF?

RESTCONF

What is it?

- It's an implementation of a REST API
- Model-driven API
- Functional sub-set of NETCONF
- Exposes YANG models via a REST API (URL)
- Uses HTTP(S) as transport
- Uses XML or JSON for encoding
- Developed to use HTTP tools and programming libraries



RESTCONF

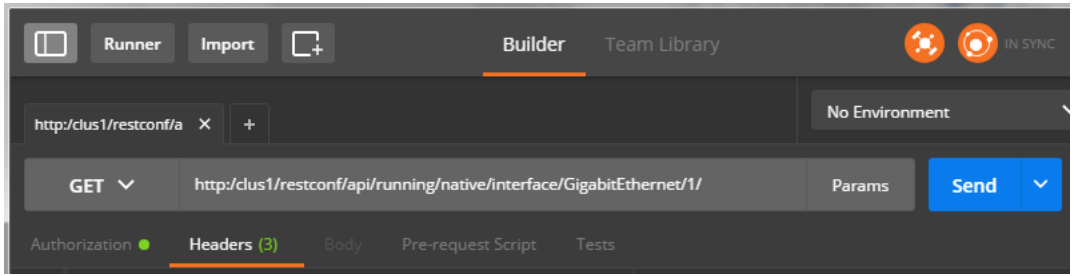
Consuming the IOS-XE RESTCONF API

- Almost identical to using any other RESTful API
- Pay attention to the **Accept** and **Content-Type** Headers
  - Typical RESTful APIs: *application/json* and *application/xml*
  - RESTCONF: *application/vnd.yang.data+json* and *application/vnd.yang.data+xml* (others exist too)

Consuming RESTful APIs

Utilities and Tools

Postman
Chrome Application



requests
Python Module

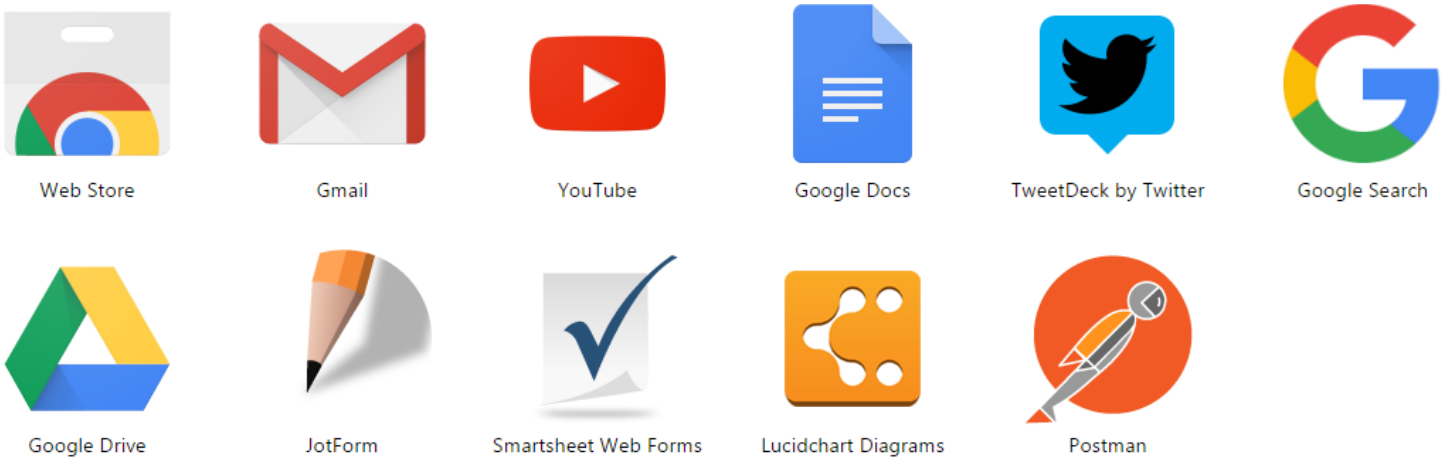
```
#!/usr/bin/env python

import requests
import json
from requests.auth import HTTPBasicAuth

if __name__ == "__main__":

    auth = HTTPBasicAuth('cisco', 'cisco')
    headers = { 'Accept': 'application/json'}
    url = 'http://clus1/restconf/api/running/native/interface/GigabitEthernet/1'
    response = requests.get(url, verify=False, headers=headers, auth=auth)
    print response.text
```

cURL
Linux Command Line tool

```
$ curl -H "Accept:application/vnd.yang.data+json" -u cisco:cisco123 http://clus1/restconf/api/running/native/interface/GigabitEthernet/1
```

What is Postman?

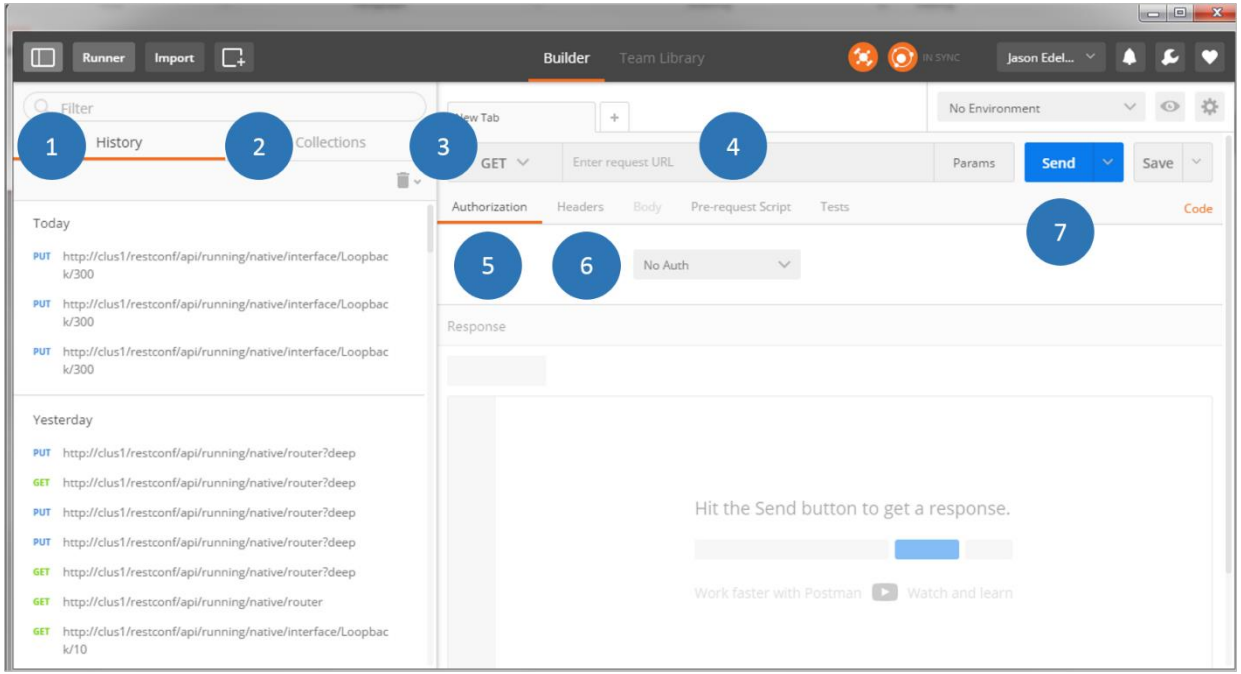
Chrome Application



Also available as a stand-alone application for Windows, MacOS or Linux.

Postman

Exploring HTTP-Based APIs



Lab 1: Exploring RESTCONF with Postman

Lab 2: Automating IOS-XE with RESTCONF using Python

Challenge 1 and 2

Diving Deeper into NETCONF

NETCONF

NETCONF is an IETF network management protocol designed specifically for configuration management

- Makes a distinction between configuration and state data
- Utilizes multiple configuration data stores (candidate, running, startup)
- Configuration change transactions
- Provides client-side configuration validation
- Uses filtering mechanisms for selective data retrieval
- Uses a client-server model and SSH as transport protocol

NETCONF

Protocol Stack

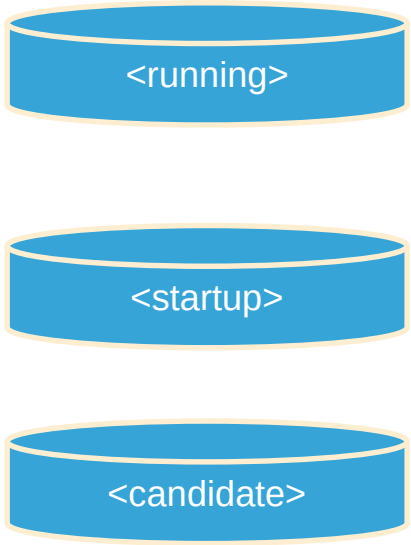
Layer	Example
Content	XML Documents (XSD, YANG, etc.)
Operations*	<get-config>, <get>, <copy-config>, <commit>, <validate>, <lock>, <unlock>, <edit-config>, <delete-config>
Messages	<rpc>,<rpc-reply>
Protocols	SSHv2, SOAP, TLS

* Varies per hardware platform and OS.

NETCONF

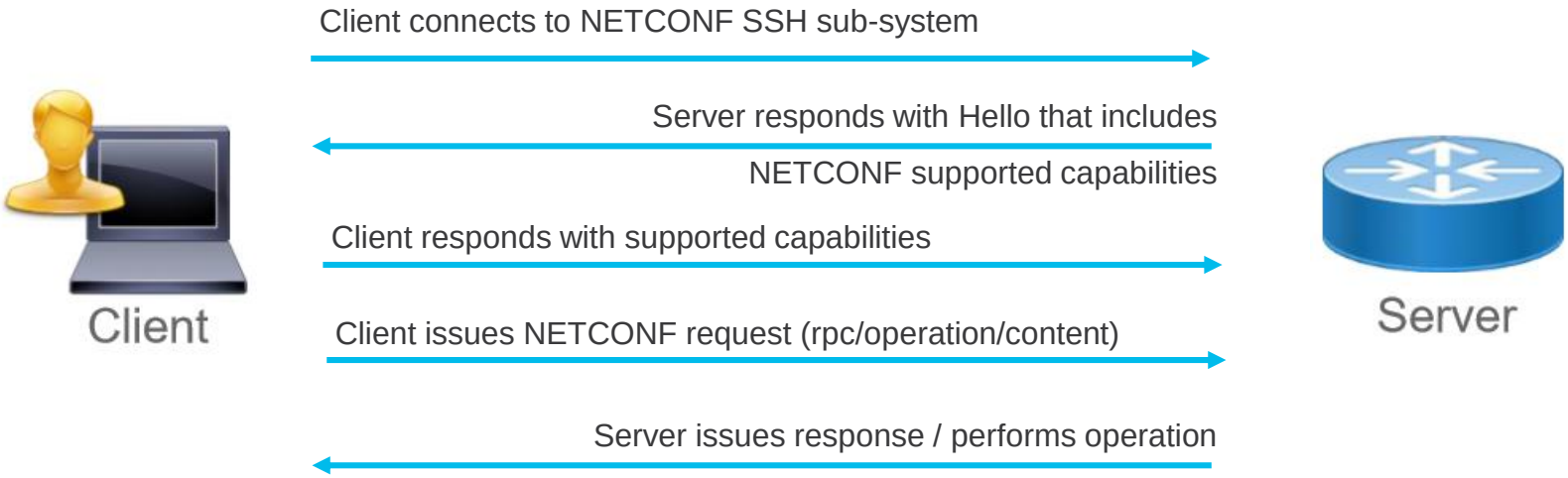
Data Stores

- Target of *Operations*
- May hold an entire copy of the configuration
- Not all data stores are supported by all devices
- Running config is the only required data store
- Not all device's are writeable
  - May have to copy from writeable one



NETCONF

NETCONF over SSH



NETCONF

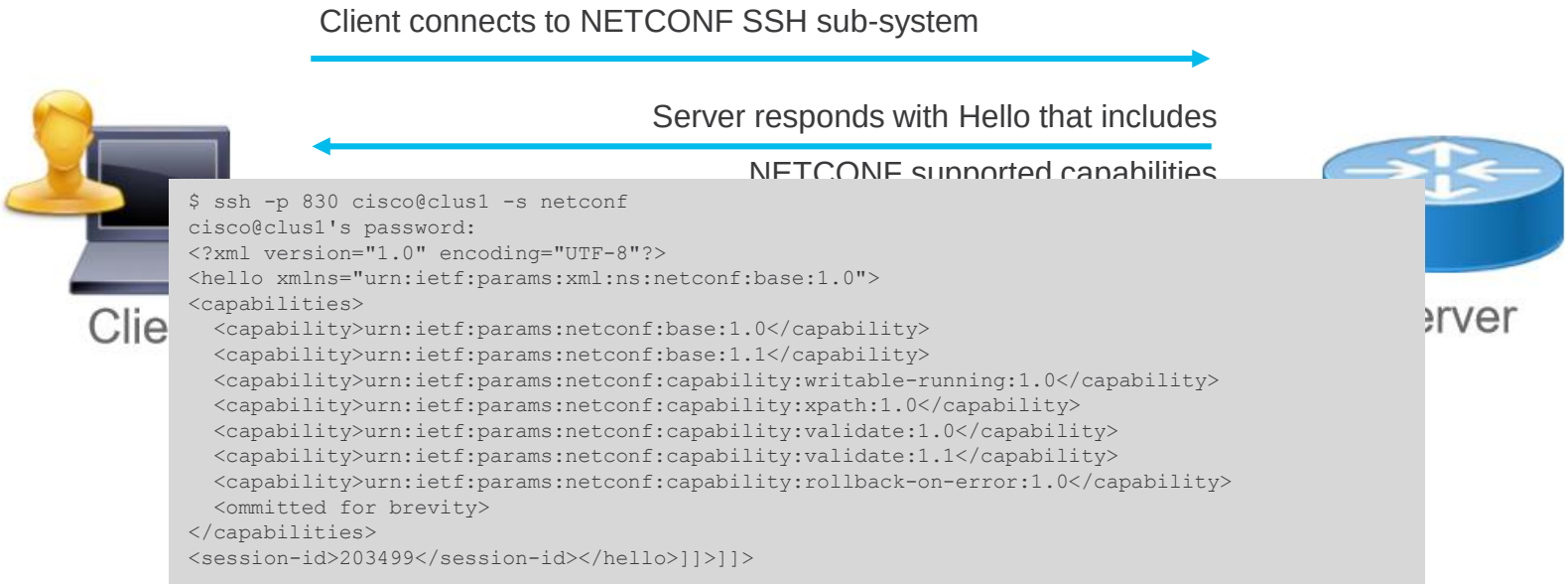
NETCONF over SSH

Client connects to NETCONF SSH sub-system

```
$ ssh -p 830 cisco@clus1 -s netconf
```

NETCONF

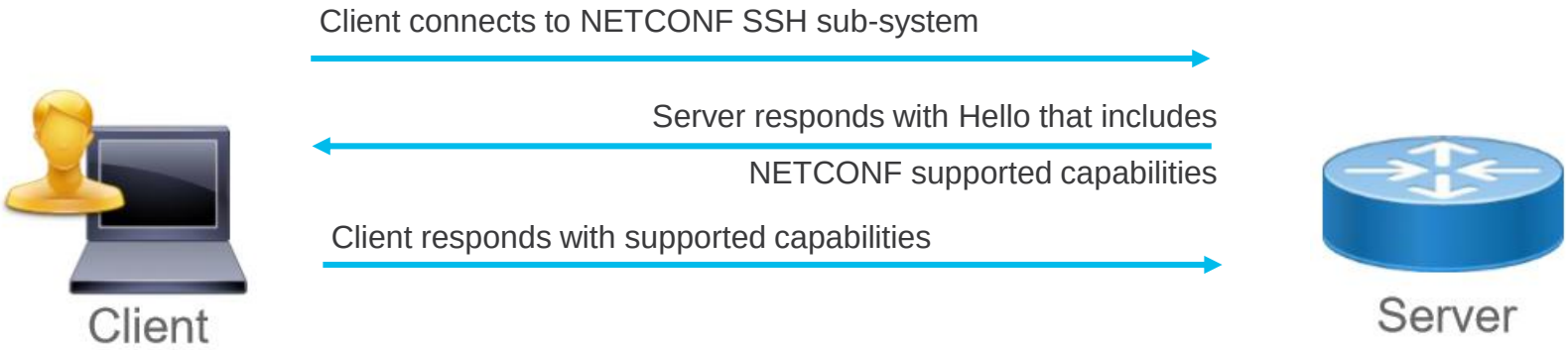
NETCONF over SSH




```
$ ssh -p 830 cisco@clus1 -s netconf
cisco@clus1's password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
  <capability>urn:ietf:params:netconf:base:1.0</capability>
  <capability>urn:ietf:params:netconf:base:1.1</capability>
  <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
  <capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
  <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
  <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
  <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
  <ommitted for brevity>
</capabilities>
<session-id>203499</session-id></hello>]]>]]>
```

NETCONF

NETCONF over SSH




```
<?xml version="1.0"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
  </capabilities>
</hello>]]>]]>
```

NETCONF

NETCONF over SSH

```
<?xml version="1.0"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter type="subtree">
            <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
             <interface>
              <GigabitEthernet>
               <name>1</name>
              </GigabitEthernet>
             </interface>
            </native>
        </filter>
    </get>
</rpc>
]]>]]>
```

Client conne

Client respc

Client

Client issues NETCONF request (rpc/operation/content)

Server

NETCONF

NETCONF over SSH

Client connects to NETCONF SSH sub-system


```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" mess
id="101"><data><native xmlns="http://cisco.com/ns/yang/Cisco-IO
XE-
native"><interface><GigabitEthernet><name>1</name><vrf><forward
MANAGEMENT</forwarding></vrf><ip><address><primary><address>10.
151</address><mask>255.255.255.0</mask></primary></address></ip
p><enabled>false</enabled></mop><negotiation
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
ethernet"><auto>true</auto></negotiation></GigabitEthernet></in
ace></native></data></rpc-reply>]]>]]>
```

Response is usually not pretty.
Use a tool to format visually.

NETCONF

Inside the Protocol Stack for IOS-XE

Layer	Example
Content	XML Documents (YANG, etc.)
Operations	<get-config>, <get>, <copy-config>, <commit>, <validate>, <lock>, <unlock>, <edit-config>, <delete-config>*
Messages	<rpc>,<rpc-reply>
Protocols	SSHv2

NETCONF

Inside the Protocol Stack for IOS-XE

Layer	Example
Content	XML Docum
Operations	<get-config> <commit>, < <unlock>, <e config>*
Messages	<rpc>,<rpc-reply>
Protocols	SSHv2

XML Encoding

```
<?xml version="1.0"?>
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
        <source>
            <url>file://candidate.cfg</url>
        </source>
        <target>
            <candidate/>
        </target>
    </copy-config>
</rpc>
]]>]]>
```

Message is <rpc>

Operation is *copy-config*

NETCONF

Inside the Protocol Stack for IOS-XE

Layer	Example
Content	XML Docum
Operations	<get-config> <commit>, < <unlock>, <e config>*
Messages	<rpc>,<rpc-reply>
Protocols	SSHv2

XML Encoding

```
<?xml version="1.0"?>
<rpc message-id="1" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <copy-config>
        <source>
            <url>file://candidate.cfg</url>
        </source>
        <target>
            <candidate/>
        <
    </cop
</rpc>
]]>]]>
```

Message is <rpc>

Operation is *copy-config*

```
<?xml version="1.0"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <get>
        <filter type="subtree">
            <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
             <interface>
              <GigabitEthernet>
               <name>1</name>
              </GigabitEthernet>
             </interface>
            </native>
        </filter>
    </get>
</rpc>
]]>]]>
```

Operation is *get*

Content is everything else inside the operation

Consuming NETCONF APIs

Utilities and Tools

SSH
NETCONF over SSH CLI

```
$ ssh -p 830 cisco@clus1 -s netconf
```

ncclient
Python Package

```
#!/usr/bin/env python

from lxml import etree
from ncclient import manager

if __name__ == "__main__":

    device = manager.connect(host='clus1', port=830, username='ntc', password='ntc123',
                             hostkey_verify=False, device_params={}, allow_agent=False,
                             look_for_keys=False)
    get_filter = """
        <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
         <interface>
          <GigabitEthernet>
           <name>1</name>
          </GigabitEthernet>
         </interface>
        </native>
    """
    nc_get_reply = device.get(('subtree', get_filter))
    print etree.tostring(nc_get_reply.data_ele, pretty_print=True)
    device.close_session()
```

Lab 3: Automating IOS-XE with NETCONF using Python

Challenge 3

Summary

- IOS-XE uses model driven APIs: RESTCONF/YANG and NETCONF/YANG APIs
- YANG models supported on IOS-XE are both Cisco and industry standard
- RESTCONF will be supported by TAC in 16.6 and later
- Use Postman to get started exploring the RESTCONF API
- Using RESTCONF is just like using any other HTTP API (remember check your headers)
- Use NETCONF if you're integrating with other devices that also support NETCONF or you need features that NETCONF offers that RESTCONF doesn't

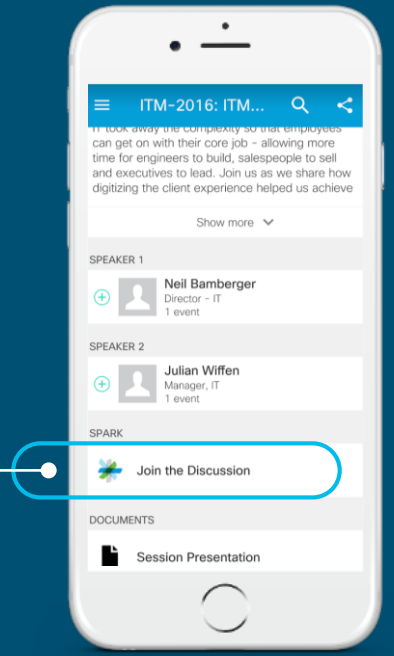
Cisco Spark

Questions?

Use Cisco Spark to communicate with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install Spark or go directly to the space
4. Enter messages/questions in the space

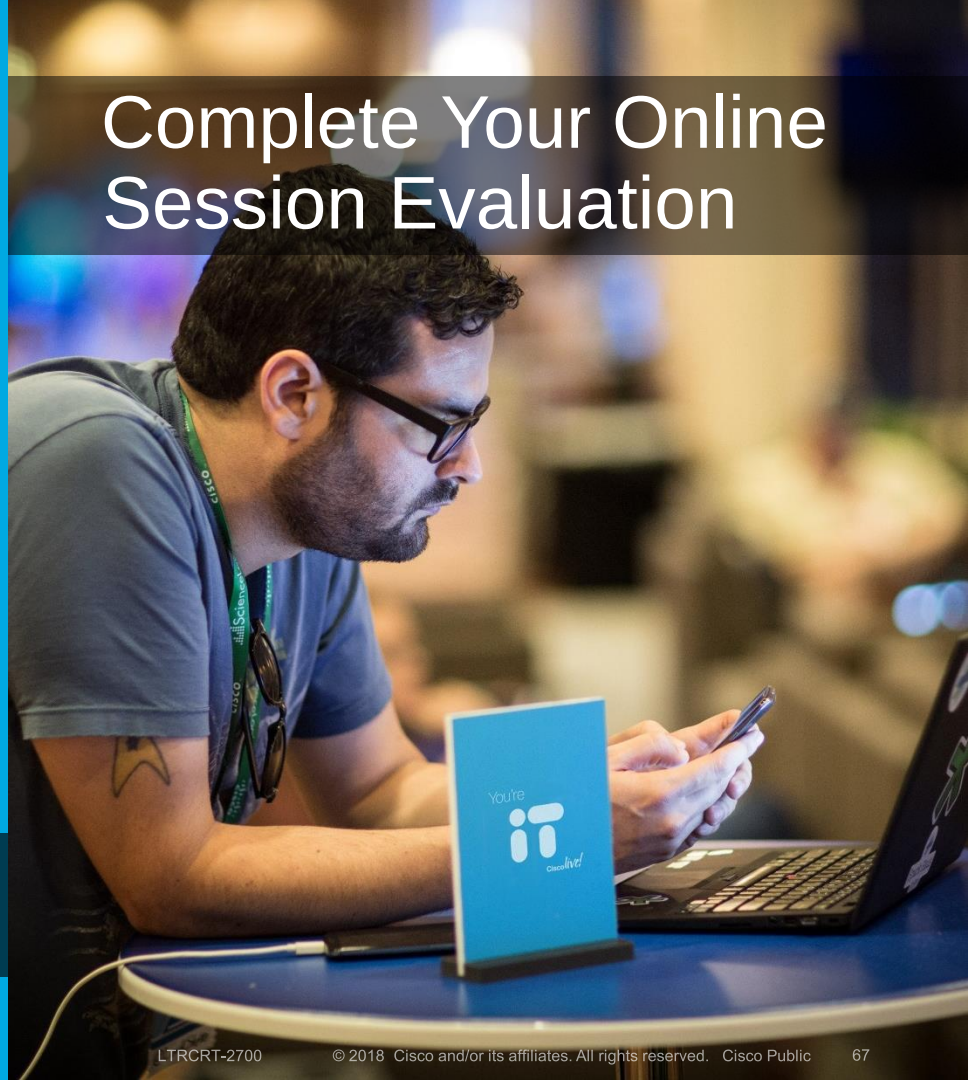


cs.co/ciscolivebot#LTRCRT-2700

Complete Your Online Session Evaluation

- Please complete your Online Session Evaluations after each session
- Complete 4 Session Evaluations & the Overall Conference Evaluation (available from Thursday) to receive your Cisco Live T-shirt
- All surveys can be completed via the Cisco Live Mobile App or the Communication Stations

Don’t forget: Cisco Live sessions will be available for viewing on-demand after the event at www.ciscolive.com/global/on-demand-library/.



Continue Your Education

- Demos in the Cisco campus
- Walk-in Self-Paced Labs
- Tech Circle
- Meet the Engineer 1:1 meetings
- Related sessions

Network Programmability Cisco Education Offerings

Course	Description	Cisco Certification
Developing with Cisco Network Programmability (NPDEV)	Provides Application Developers with comprehensive curriculum to develop infrastructure programming skills; Addresses needs of software engineers who automate network infrastructure and/or utilize APIs and toolkits to interface with SDN controllers and individual devices	Cisco Network Programmability Developer (NPDEV) Specialist Certification
Designing and Implementing Cisco Network Programmability (NPDESI)	Provides network engineers with comprehensive soup-to-nuts curriculum to develop and validate automation and programming skills; Directly addresses the evolving role of network engineers towards more programmability, automation and orchestration	Cisco Network Programmability Design and Implementation (NPDESI) Specialist Certification
Programming for Network Engineers (PRNE)	Learn the fundamentals of Python programming – within the context of performing functions relevant to network engineers. Use Network Programming to simplify or automate tasks	Recommended pre-requisite for NPDESI and NPDEV Specialist Certifications
Cisco Digital Network Architecture Implementation Essentials (DNAIE)	This training provides students with the guiding principles and core elements of Cisco's Digital Network Architecture (DNA) architecture and its solution components including; APIC-EM, NFV, Analytics, Security and Fabric.	None

For more details, please visit: http://learningnetwork.cisco.com

Thank you


You're
Cisco live!