Network Programmability YANG/NETCONF/RESTCONF

Cisco DevNet Webinar Series

Speaker: Hank Preston III| DevNet Developer Evangelist
Hostess: Kara Sullivan | Cisco Networking Academy

15 March 2018

Welcome to the 7th session of the Cisco DevNet webinar series

- Use the Q and A panel to ask questions.
- Use the Chat panel to communicate with attendees and panelists.
- A link to a recording of the session will be sent to all registered attendees.
- Please take the feedback survey at the end of the webinar.

Cisco DevNet Series

1 Intro to Software & Programmability

2 Intro to Coding

3 Intent Networks: How to be a Network Engineer in a Programmable Age

4 Fast Lane: Where Code (Apple) Meets Network Infrastructure (Cisco)

5 APIs with Cisco Spark

6 Network Programmability & APIC-EM

7 Network Programmability with YANG/NETCONF/RESTCONF – Today!

All Series Details can be Found @ http://bit.ly/devnetseries

Joining You Today:

Hank Preston III

Developer Evangelist

DevNet, Cisco

CISCO DEVNET

Introduction to Model Driven Programmability

Breaking down YANG, NETCONF, and RESTCONF

Hank Preston, ccie 38336 R/S
NetDevOps Evangelist
@hfpreston

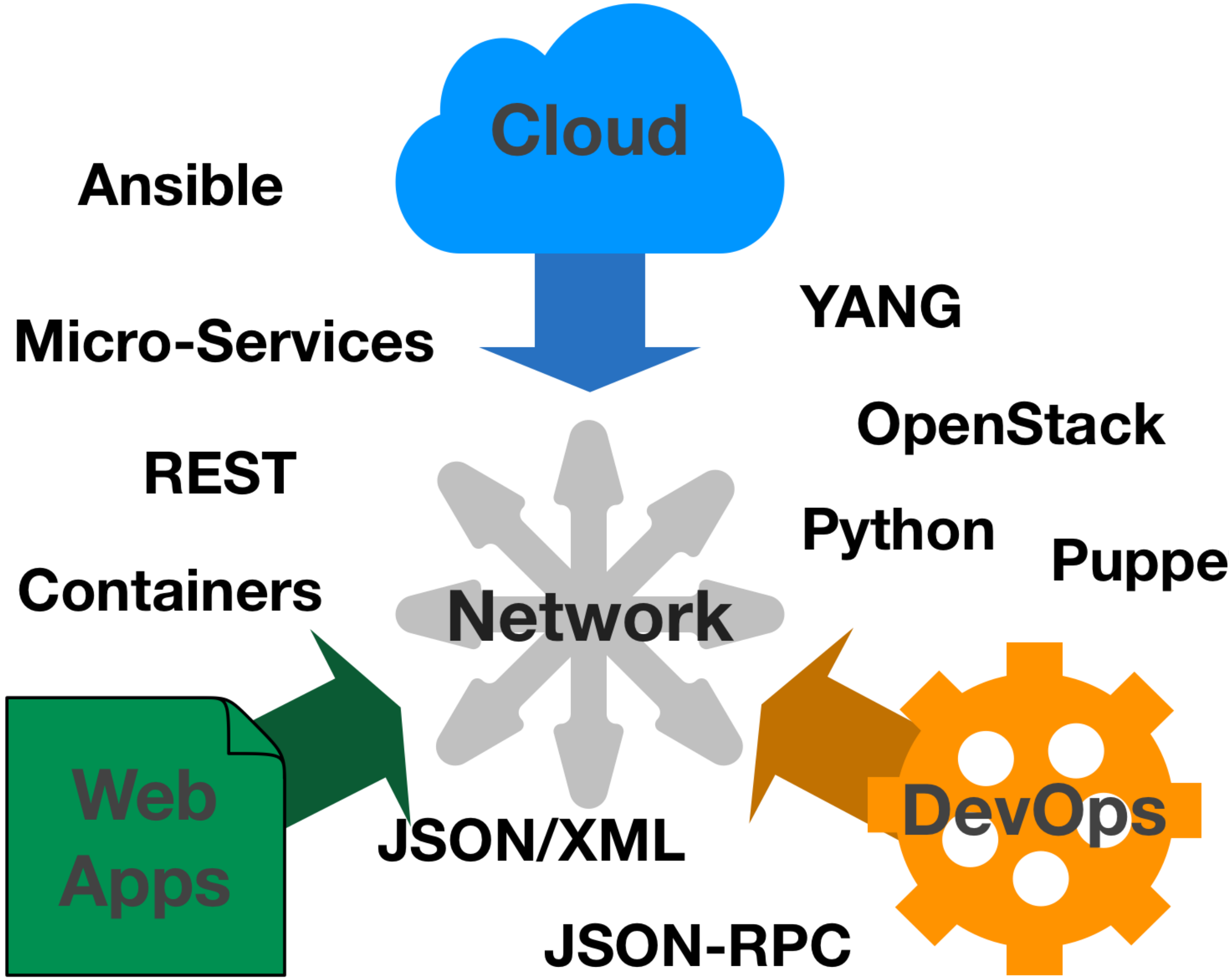
Agenda

- The Road to Model Driven Programmability
- Introduction to YANG Data Models
- Introduction to NETCONF
- Introduction to RESTCONF
- Conclusion and Q/A

Note: All code samples referenced in this presentation are available at
https://github.com/CiscoDevNet/BRKDEV-1368

The Road to Model Driven Programmability

The Network is No Longer Isolated



Ansible

Micro-Services

REST

Containers

YANG

OpenStack

Python

Puppet

JSON/XML

JSON-RPC

What about SNMP?

SNMP works "reasonably well for device monitoring"

RFC 3535: Overview of the 2002 IAB Network Management Workshop – 2003
https://tools.ietf.org/html/rfc3535

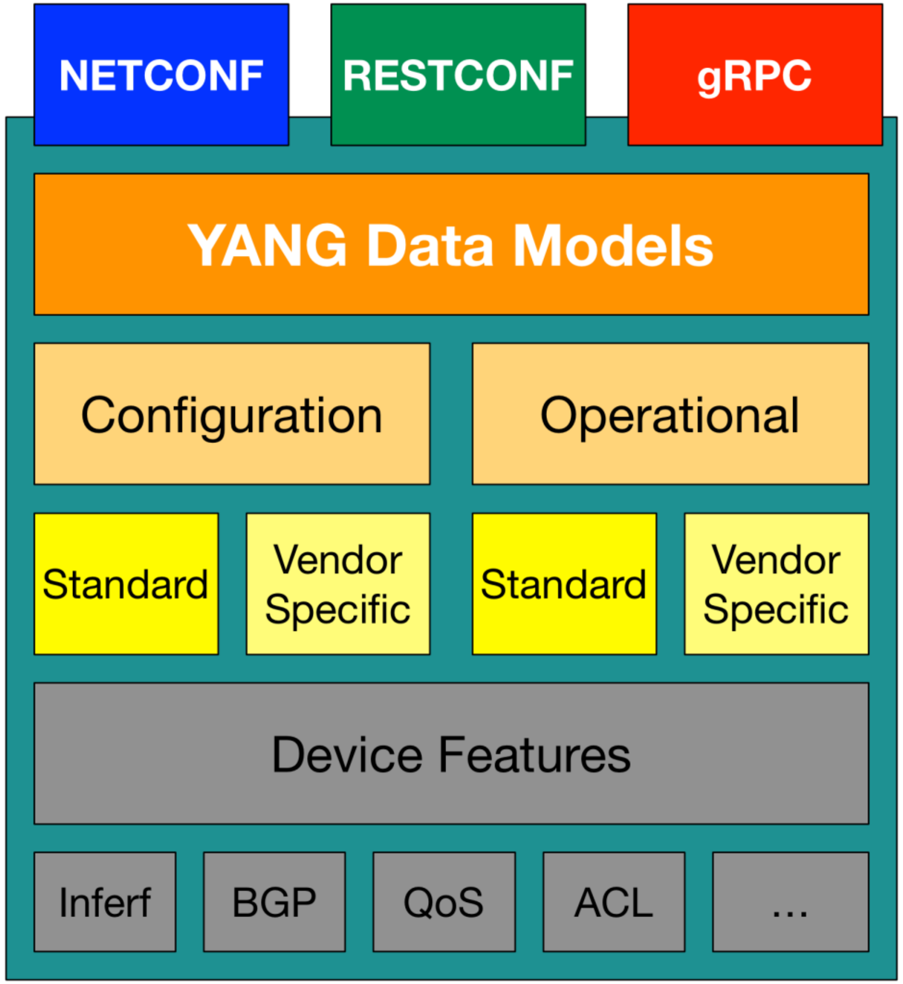
- Typical config: SNMPv2 read-only community strings
- Typical usage: interface statistics queries and traps
- Empirical Observation: SNMP is not used for configuration
  - Lack of Writeable MIBs
  - Security Concerns
  - Difficult to Replay/Rollback
  - Special Applications

RFC 3535: What is Needed?

- A programmatic interface for device configuration
- Separation of Configuration and State Data
- Ability to configure " services" NOT " devices"
- Integrated error checking and recovery

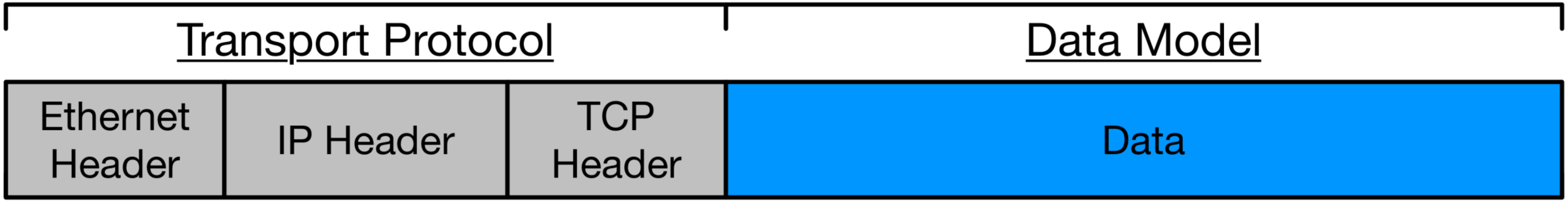
Model Driven Programmability

- NETCONF – 2006 – RFC 4741 (RFC 6241 in 2011)
- YANG – 2010 – RFC 6020
- RESTCONF – 2017 – RFC 8040
- gRPC – 2015 – OpenSource project by Google
  - *Not covered in today's session*



Transport (Protocol) vs Data (Model)

TCP/IP Network Frame Format



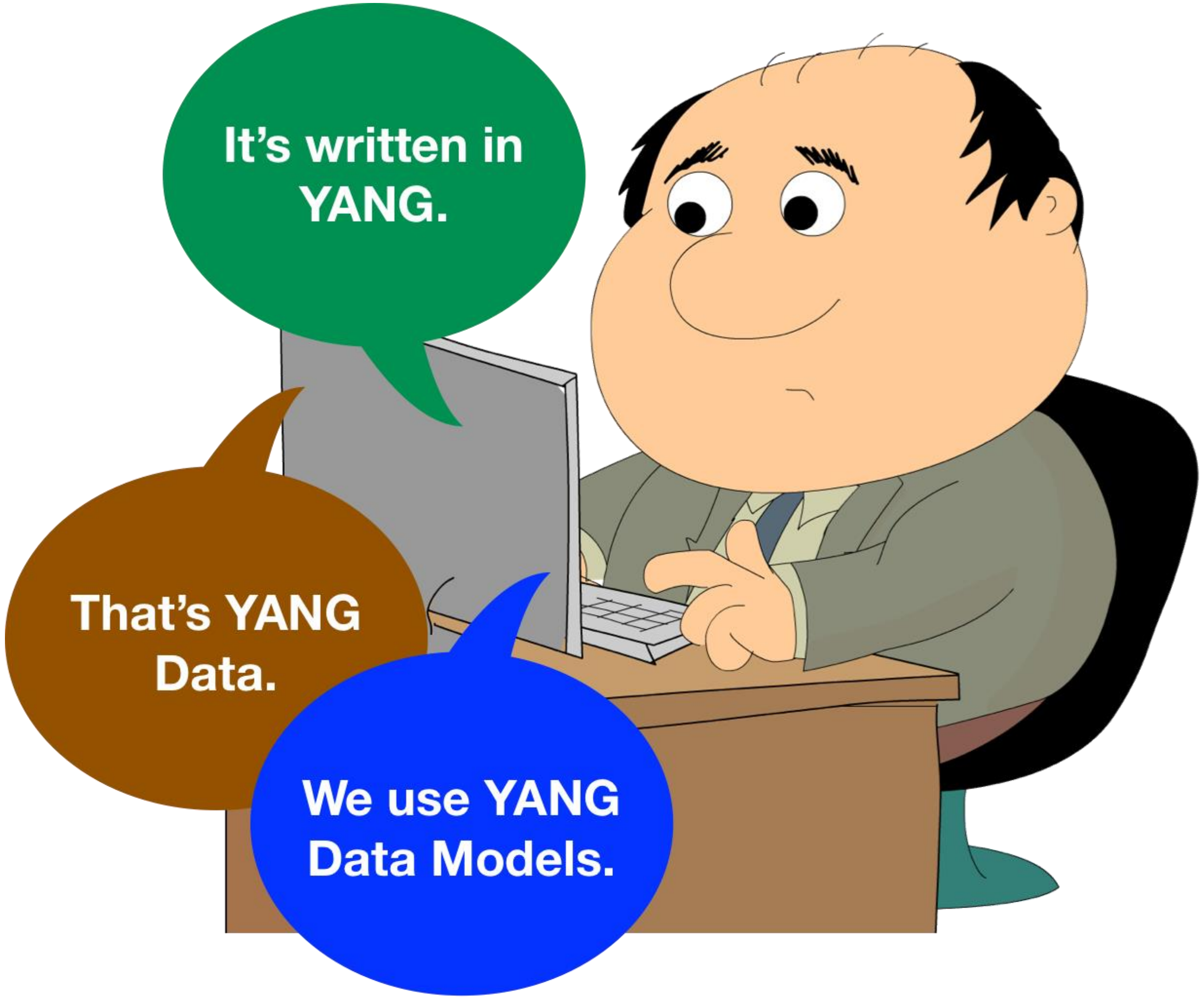
Transport Protocol			Data Model
Ethernet Header	IP Header	TCP Header	Data

- NETCONF
- RESTCONF
- gRPC

- YANG

What is YANG?

Three Meanings of “YANG”



YANG Modeling Language

- Module that is a self-contained top-level hierarchy of nodes
- Uses containers to group related nodes
- Lists to identify nodes that are stored in sequence
- Each individual attribute of a node is represented by a leaf
- Every leaf must have an associated type

```
module ietf-interfaces {
  import ietf-yang-types {
    prefix yang;
  }
  container interfaces {
    list interface {
      key "name";
      leaf name {
        type string;
      }
      leaf enabled {
        type boolean;
        default "true";
    }
}
```

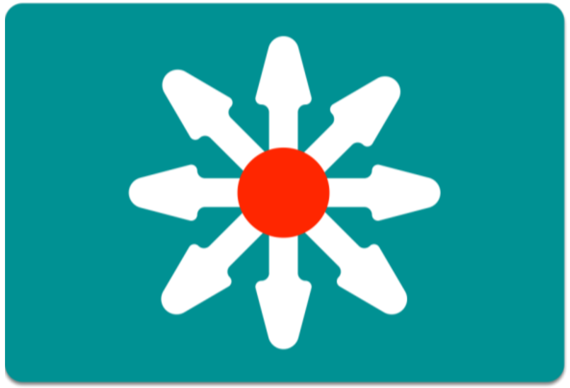
Example edited for simplicity and brevity

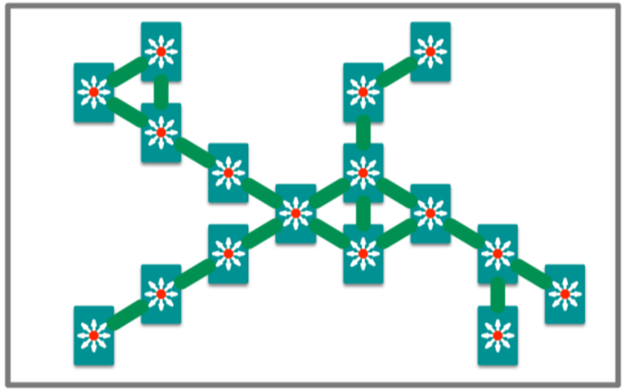
What is a Data Model?

A data model is simply a well understood and agreed upon method to describe " something" . As an example, consider this simple " data model" for a person.

- *Person*
  - Gender – male, female, other
  - Height – Feet/Inches or Meters
  - Weight – Pounds or Kilos
  - Hair Color – Brown, Blond, Black, Red, other
  - Eye Color – Brown, Blue, Green, Hazel, other

What might a YANG Data Model describe?





Device Data Models

- Interface
- VLAN
- Device ACL
- Tunnel
- OSPF
- etc

Service Data Models

- L3 MPLS VPN
- MP-BGP
- VRF
- Network ACL
- System Management
- Network Faults
- etc

Working with YANG Data Models

Where do Models Come From?

Industry Standard

- **Standard definition**
  (IETF, ITU, OpenConfig, etc.)

- Compliant with standard

```
ietf-diffserv-policy.yang
ietf-diffserv-classifer.yang
ietf-diffserv-target.yang
```

Vendor Specific

- **Vendor definition**
  (i.e. Cisco)

- Unique to Vendor Platforms

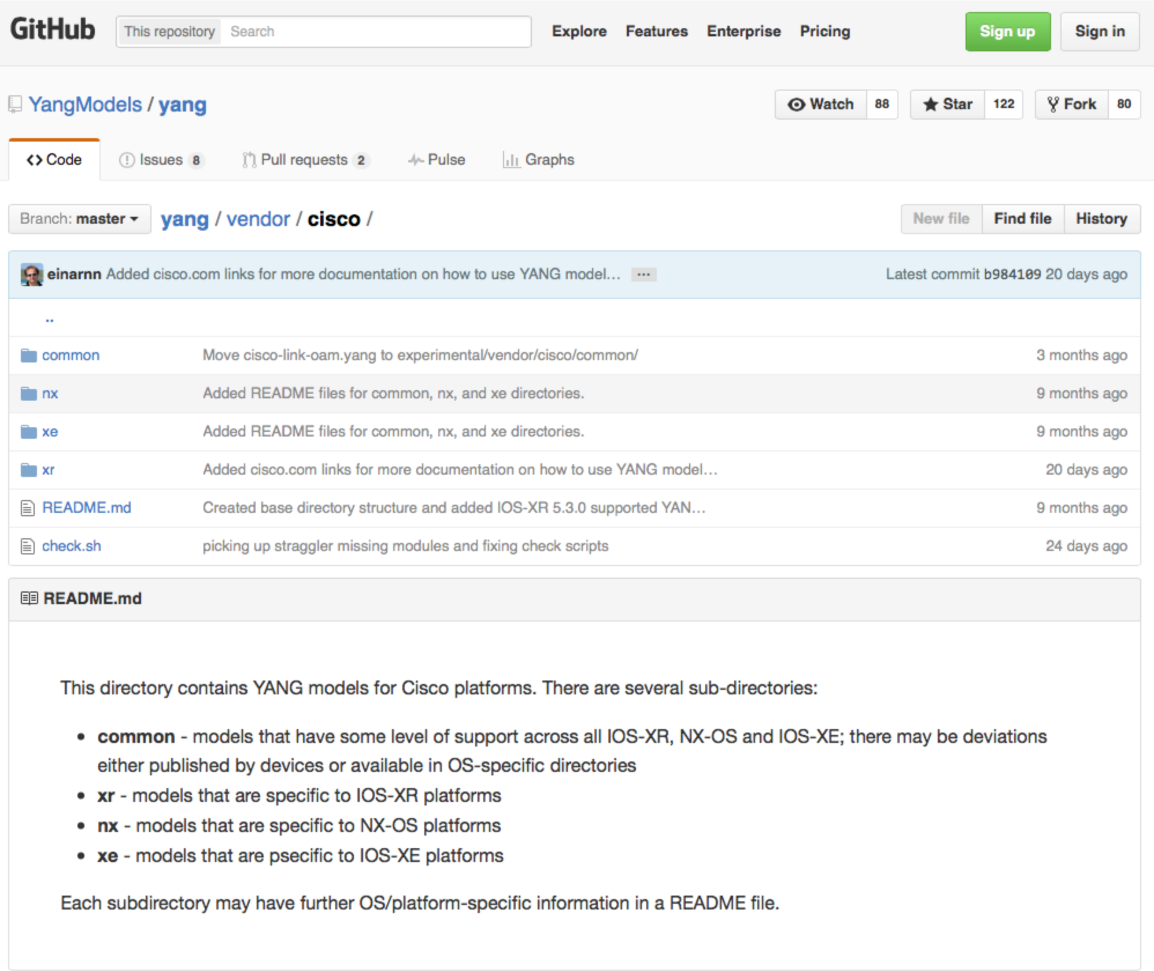
```
cisco-memory-stats.yang
cisco-flow-monitor
cisco-qos-action-qlimit-cfg
```

https://github.com/YangModels/yang

Where to get the Models?

- For YANG modules from standard organizations such as the IETF, open source such as Open Daylight or vendor specific modules”
  - https://github.com/YangModels/yang

- For OpenConfig models
  - https://github.com/openconfig/public



einarnn Added cisco.com links for more documentation on how to use YANG model… Latest commit b984109 20 days ago

..		
common	Move cisco-link-oam.yang to experimental/vendor/cisco/common/	3 months ago
nx	Added README files for common, nx, and xe directories.	9 months ago
xe	Added README files for common, nx, and xe directories.	9 months ago
xr	Added cisco.com links for more documentation on how to use YANG model…	20 days ago
README.md	Created base directory structure and added IOS-XR 5.3.0 supported YAN…	9 months ago
check.sh	picking up straggler missing modules and fixing check scripts	24 days ago

README.md

This directory contains YANG models for Cisco platforms. There are several sub-directories:

- **common** - models that have some level of support across all IOS-XR, NX-OS and IOS-XE; there may be deviations either published by devices or available in OS-specific directories
- **xr** - models that are specific to IOS-XR platforms
- **nx** - models that are specific to NX-OS platforms
- **xe** - models that are psecific to IOS-XE platforms

Each subdirectory may have further OS/platform-specific information in a README file.

YANG Data Models

The model can be displayed and represented in any number of formats depending on needs at the time. Some options include:

- YANG Language
- Clear Text
- XML
- JSON
- HTML/JavaScript

Working with YANG Models

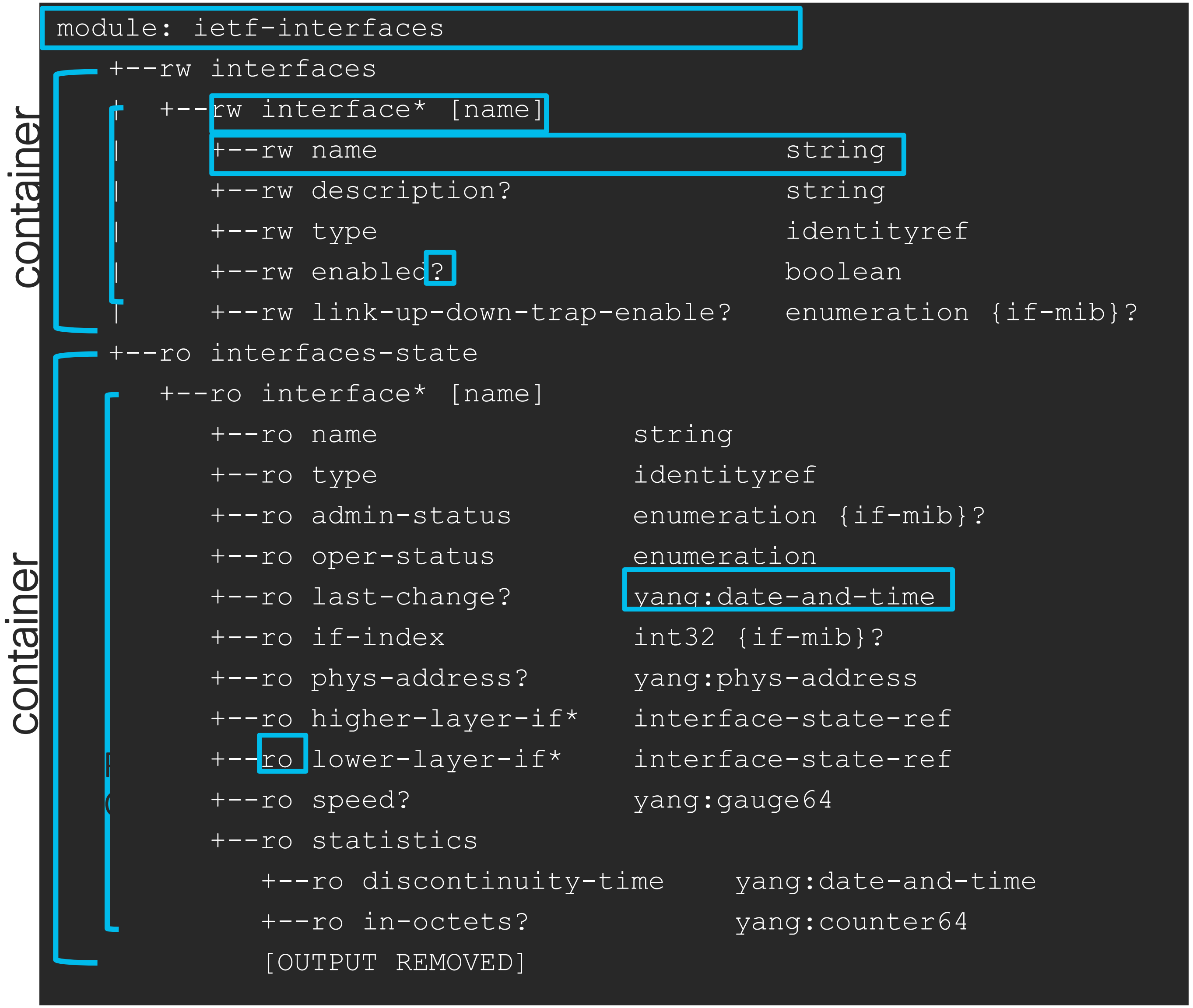
```
DevNet$ pyang -f tree ietf-interfaces.yang

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}?
```

Example output edited for simplicity and brevity

BRKDEV-1368/yang/ietf-interfaces.yang

Using pyang

- Python YANG Library
- Validate and display YANG files
- Many formats for display
  - Text: tree
  - HTML: jstree



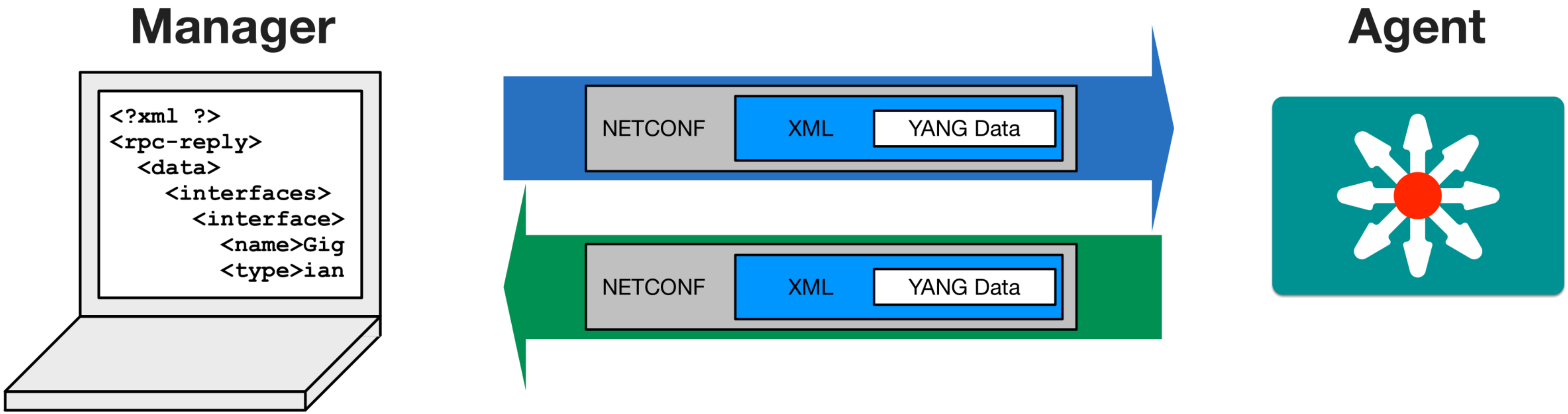
```
module: ietf-interfaces
   +--rw interfaces
   |  +--rw interface* [name]
   |     +--rw name                        string
   |     +--rw description?                string
   |     +--rw type                        identityref
   |     +--rw enabled?                    boolean
   |     +--rw link-up-down-trap-enable?   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
         +--ro last-change?       yang:date-and-time
         +--ro if-index           int32 {if-mib}?
         +--ro phys-address?      yang:phys-address
         +--ro higher-layer-if*   interface-state-ref
         +--ro lower-layer-if*    interface-state-ref
         +--ro speed?             yang:gauge64
         +--ro statistics
            +--ro discontinuity-time    yang:date-and-time
            +--ro in-octets?            yang:counter64
            [OUTPUT REMOVED]
```

Example edited for simplicity and brevity

Network Device Data in YANG

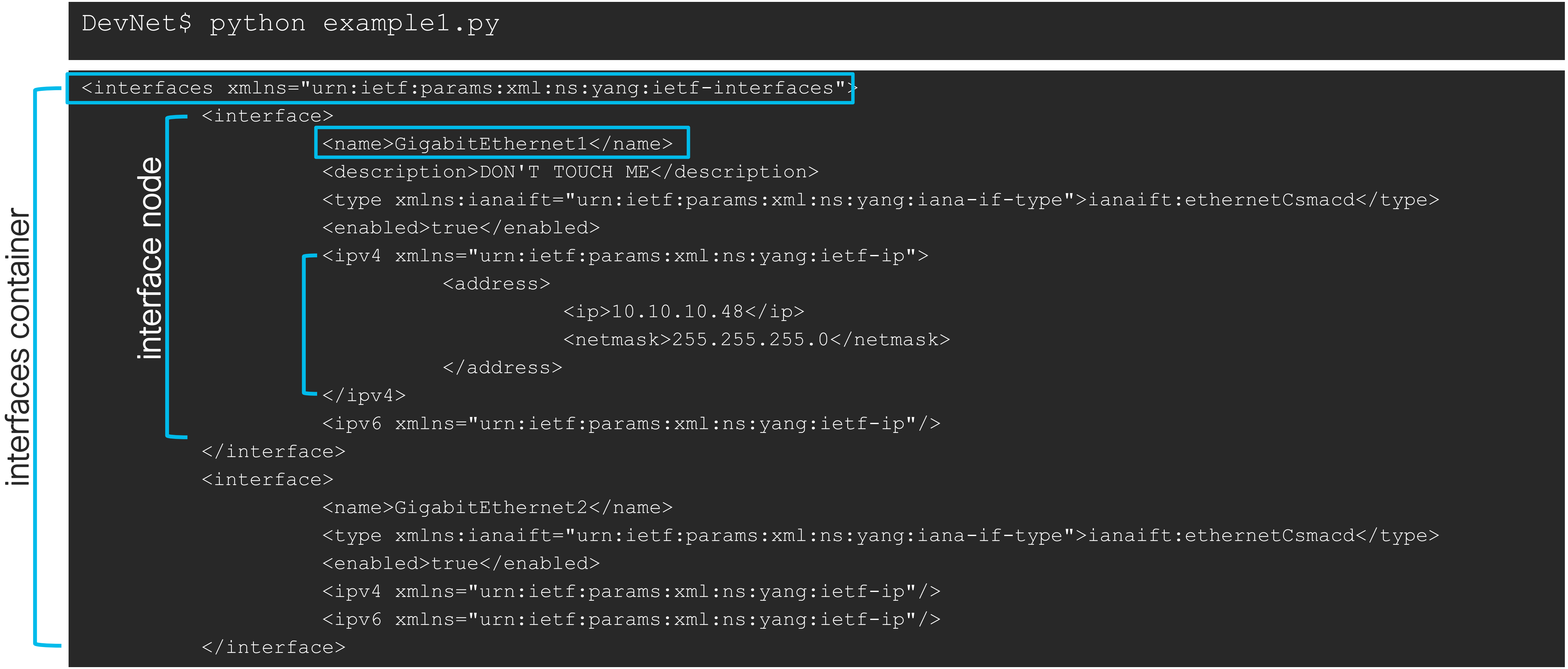
Actual Device Data Modeled in YANG

NETCONF Communications



```
<?xml ?>
<rpc-reply>
  <data>
    <interfaces>
      <interface>
        <name>Gig
        <type>ian
```

Use NETCONF to Retrieve ietf-interfaces data



```
DevNet$ python example1.py
```

```
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
          <interface>
                    <name>GigabitEthernet1</name>
                    <description>DON'T TOUCH ME</description>
                    <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
                    <enabled>true</enabled>
                    <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                              <address>
                                        <ip>10.10.10.48</ip>
                                        <netmask>255.255.255.0</netmask>
                              </address>
                    </ipv4>
                    <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
          </interface>
          <interface>
                    <name>GigabitEthernet2</name>
                    <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
                    <enabled>true</enabled>
                    <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
                    <ipv6 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip"/>
          </interface>
```

BRKDEV-1368/yang/device_info.py
BRKDEV-1368/yang/example1.py

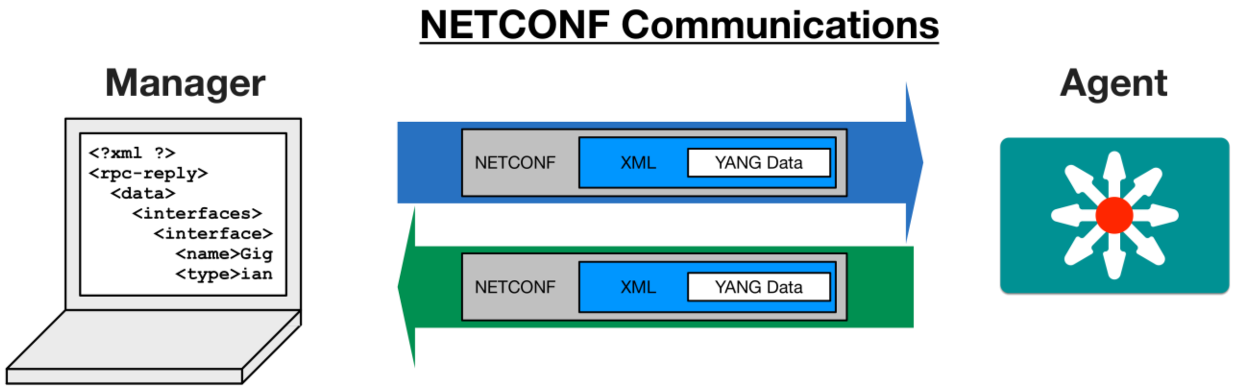
YANG Summary

YANG Summary

- YANG is a Data Modeling Language
- YANG Modules are constructed to create standard data models for network data
- YANG Data sent to or from a network device will be formatted in either XML or JSON depending on the protocol (ex: NETCONF or RESTCONF)

Understanding NETCONF

Introducing the NETCONF Protocol

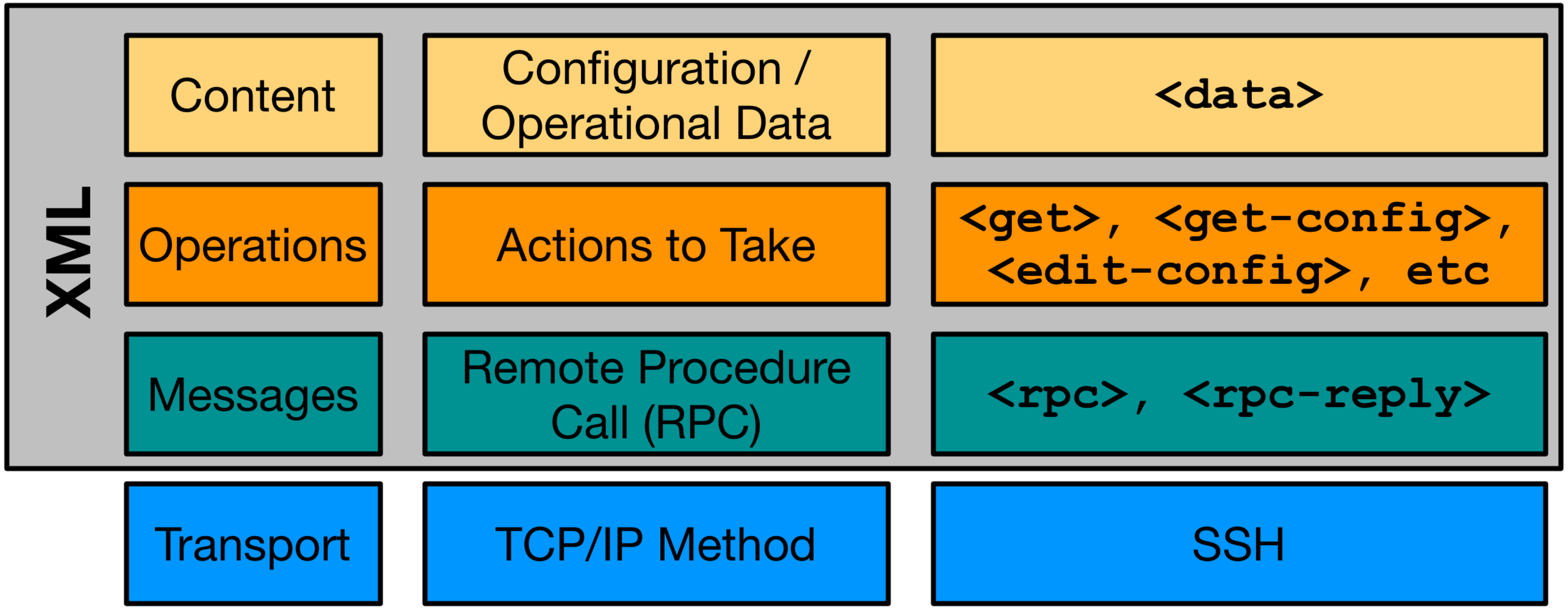


```
<?xml ?>
<rpc-reply>
  <data>
    <interfaces>
      <interface>
        <name>Gig
        <type>ian
```

Some key details:

- Initial standard in 2006 with RFC4741
- Latest standard is RFC6241 in 2011
- Does **NOT** explicitly define content

NETCONF Protocol Stack



XML	Content	Configuration / Operational Data	`<data>`
	Operations	Actions to Take	`<get>, <get-config>, <edit-config>, etc`
	Messages	Remote Procedure Call (RPC)	`<rpc>, <rpc-reply>`
	Transport	TCP/IP Method	SSH

Transport - SSH

```
$ ssh admin@192.168.0.1 -p 830 -s netconf
admin@192.168.0.1's password:
```

SSH Login

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
  <capability>urn:ietf:params:netconf:base:1.1</capability>
  <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
  <capability>urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring</capability>
  <capability>urn:ietf:params:xml:ns:yang:ietf-interfaces</capability>
  [output omitted and edited for clarity]
</capabilities>
<session-id>19150</session-id></hello>]]>]]>
```

Server (Agent) sends hello

```
<?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>
</capabilities>
  </hello>]]>]]>
```

Client (Manager) sends hello

Example edited for simplicity and brevity

Transport - SSH

```
$ ssh admin@192.168.0.1 -p 830 -s netconf
admin@192.168.0.1's password:
```

SSH Login

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
  <capability>urn:ietf:params:netconf:base:1.1</capability>
  <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
  <capability>urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring</capability>
  <capability>urn:ietf:params:xml:ns:yang:ietf-interfaces</capability>
  [output omitted and edited for clarity]
</capabilities>
<session-id>19150</session-id></hello>]]>]]>
```

Don’t NETCONF Like this!

Server (Agent) sends hello

```
<?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>
</capabilities>
  </hello>]]>]]>
```

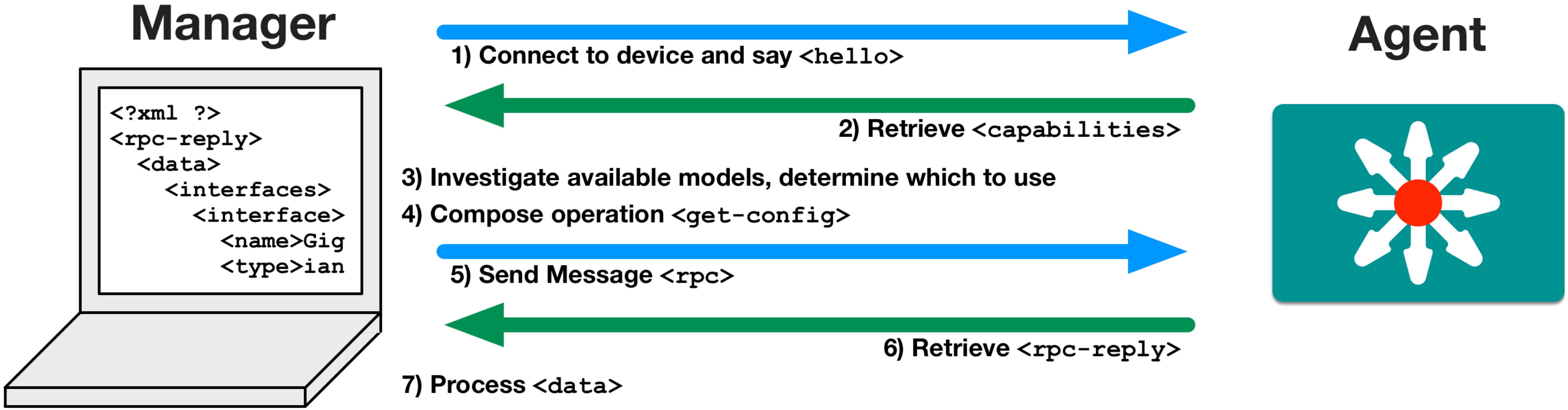
Client (Manager) sends hello

Example edited for simplicity and brevity

Operations - NETCONF Actions

Operation	Description
`<get>`	Retrieve running configuration and device state information
`<get-config>`	Retrieve all or part of specified configuration data store
`<edit-config>`	Loads all or part of a configuration to the specified configuration data store
`<copy-config>`	Replace an entire configuration data store with another
`<delete-config>`	Delete a configuration data store
`<commit>`	Copy candidate data store to running data store
`<lock> / <unlock>`	Lock or unlock the entire configuration data store system
`<close-session>`	Graceful termination of NETCONF session
`<kill-session>`	Forced termination of NETCONF session

NETCONF Communications



Manager

```
<?xml ?>
<rpc-reply>
  <data>
    <interfaces>
      <interface>
        <name>Gig
        <type>ian
```

Agent

1) Connect to device and say `<hello>`

2) Retrieve `<capabilities>`

3) Investigate available models, determine which to use

4) Compose operation `<get-config>`

5) Send Message `<rpc>`

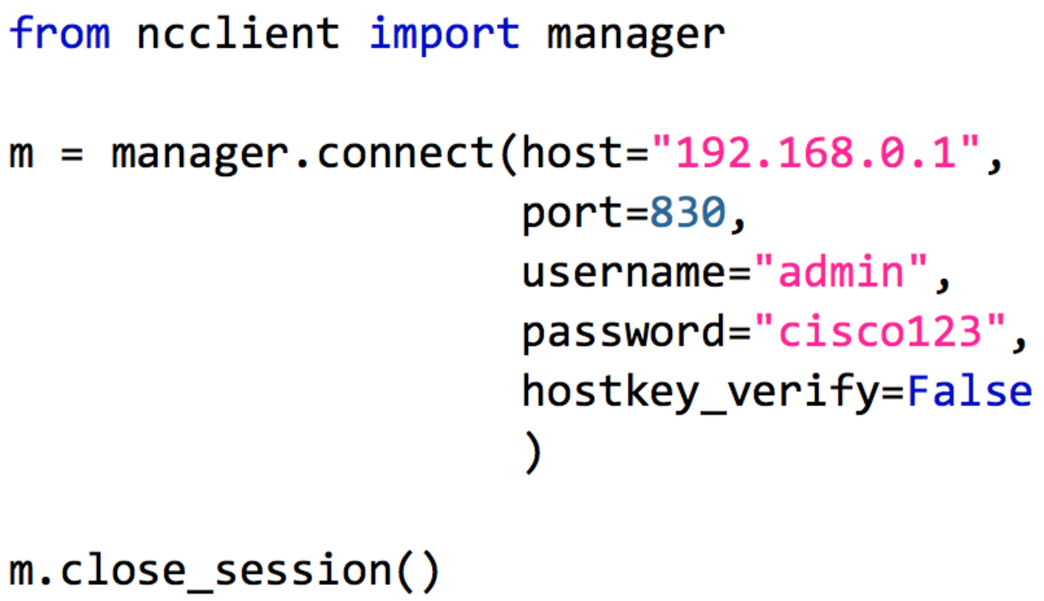
6) Retrieve `<rpc-reply>`

7) Process `<data>`

NETCONF in Code with Python

NETCONF and Python: ncclient

- Full NETCONF Manager implementation in Python
  - https://ncclient.readthedocs.io
- Simplifies connection and communication.
- Deals in raw XML

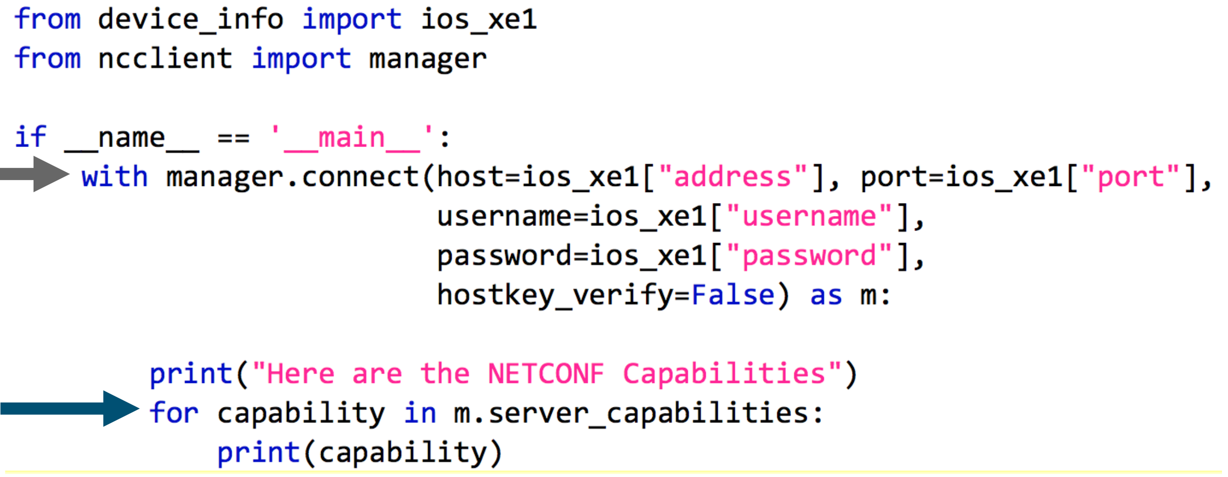


```
from ncclient import manager

m = manager.connect(host="192.168.0.1",
                    port=830,
                    username="admin",
                    password="cisco123",
                    hostkey_verify=False
                    )

m.close_session()
```

From: http://ncclient.readthedocs.io/en/latest/

Saying <hello> with Python and ncclient

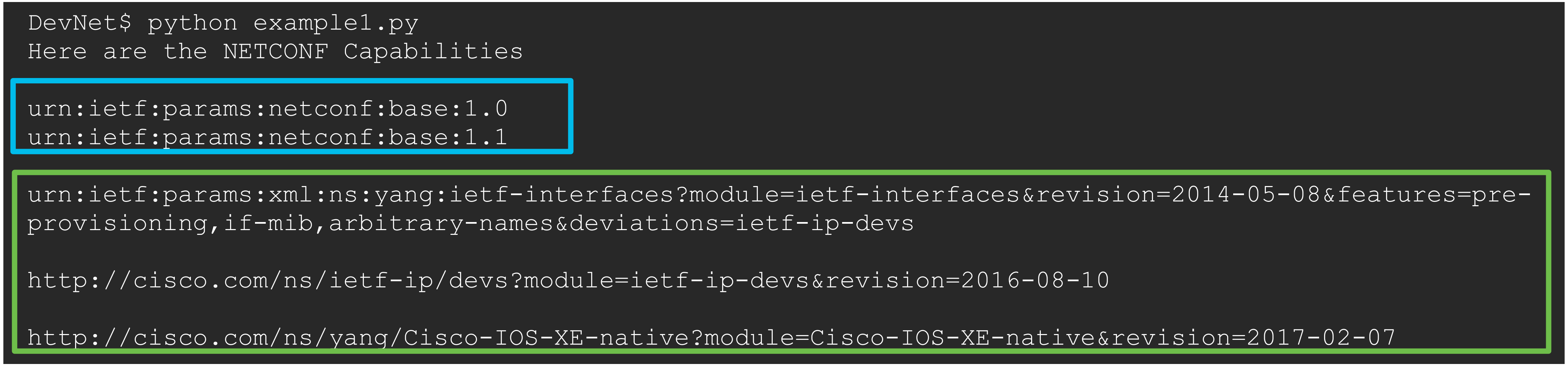
- example1.py: Saying `<hello>`
- `manager.connect()` opens NETCONF session with device
  - Parameters: host & port, user & password
  - `hostkey_verify=False` Trust cert
- Stores capabilities



```
from device_info import ios_xe1
from ncclient import manager

if __name__ == '__main__':
    with manager.connect(host=ios_xe1["address"], port=ios_xe1["port"],
                         username=ios_xe1["username"],
                         password=ios_xe1["password"],
                         hostkey_verify=False) as m:

        print("Here are the NETCONF Capabilities")
        for capability in m.server_capabilities:
            print(capability)
```

BRKDEV-1368/netconf/device_info.py
BRKDEV-1368/netconf/example1.py

Understanding the Capabilities List



```
DevNet$ python example1.py
Here are the NETCONF Capabilities

urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1

urn:ietf:params:xml:ns:yang:ietf-interfaces?module=ietf-interfaces&revision=2014-05-08&features=pre-
provisioning,if-mib,arbitrary-names&deviations=ietf-ip-devs

http://cisco.com/ns/ietf-ip/devs?module=ietf-ip-devs&revision=2016-08-10

http://cisco.com/ns/yang/Cisco-IOS-XE-native?module=Cisco-IOS-XE-native&revision=2017-02-07
```

Example edited for simplicity and brevity

Two General Types

- Base NETCONF capabilities
- Data Models Supported

Understanding the Capabilities List

```
urn:ietf:params:xml:ns:yang:ietf-interfaces
   ? module=ietf-interfaces
   & revision=2014-05-08
   & features=pre-provisioning,if-mib,arbitrary-names
   & deviations=ietf-ip-devs
.
http://cisco.com/ns/ietf-ip/devs
   ? module=ietf-ip-devs
   & revision=2016-08-10
```

Example edited for simplicity and brevity

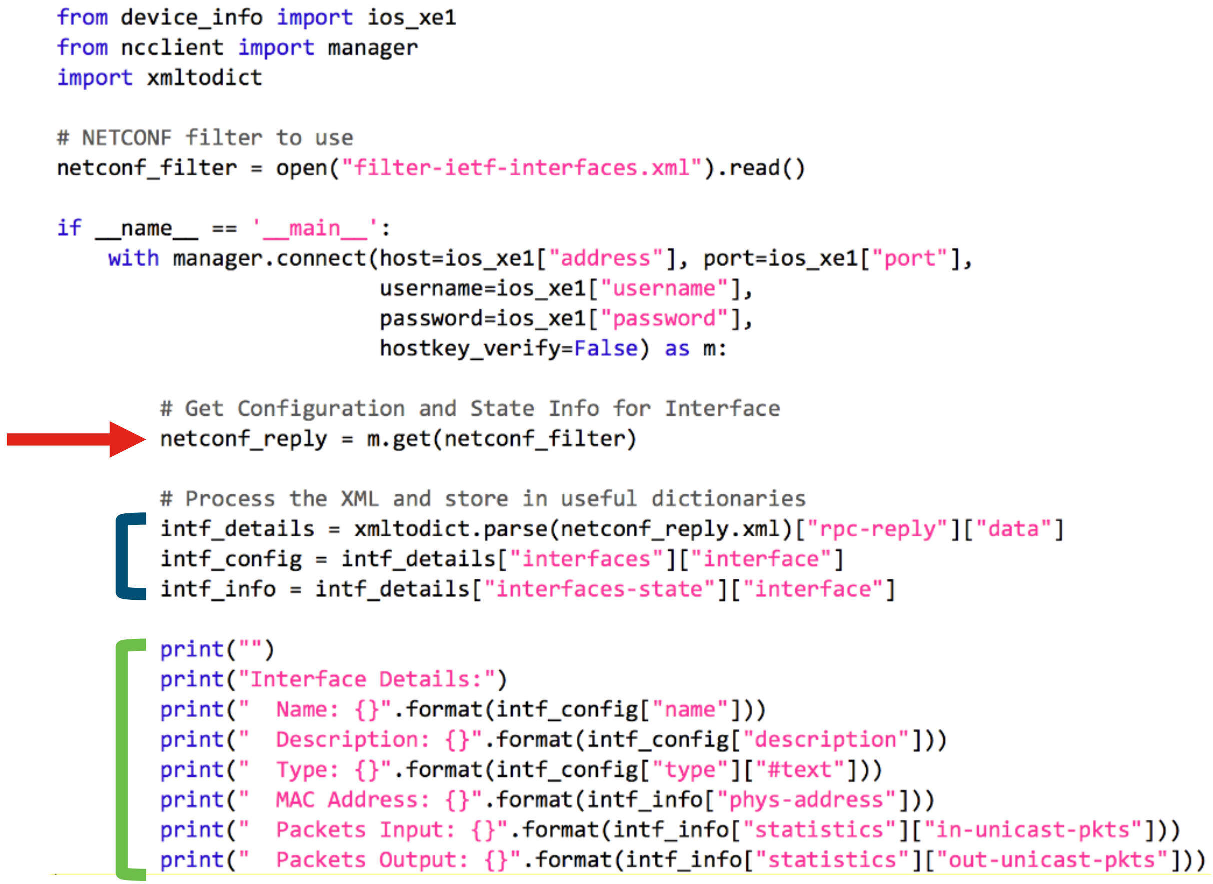
Data Model Details

- Model URI
- Module Name and Revision Date
- Protocol Features
- Deviations – Another model that modifies this one

Automate Your Network with NETCONF

Getting Interface Details with XML Filter

- example2.py: Retrieving info with ncclient
- Send <get> to retrieve config and state data
- Process and leverage XML within Python
- Report back current state of interface



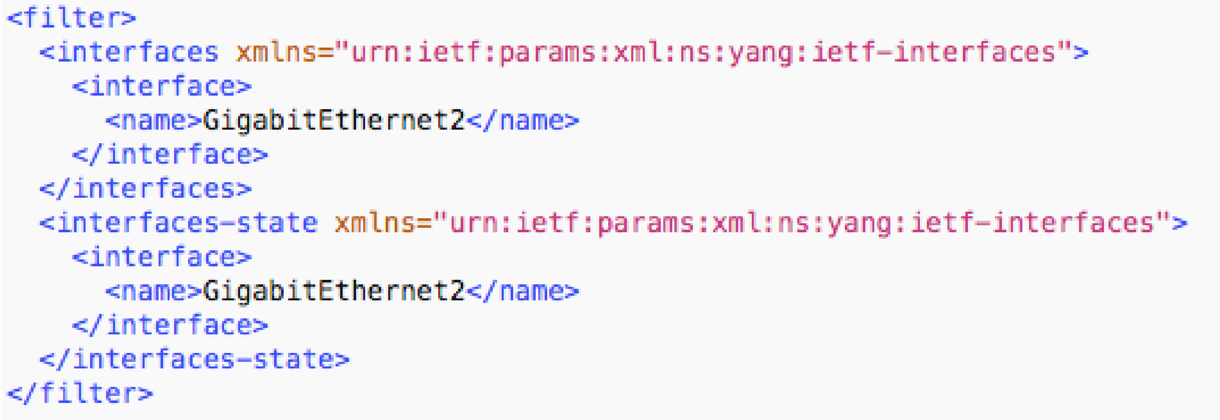
```python
from device_info import ios_xe1
from ncclient import manager
import xmltodict

# NETCONF filter to use
netconf_filter = open("filter-ietf-interfaces.xml").read()

if __name__ == '__main__':
    with manager.connect(host=ios_xe1["address"], port=ios_xe1["port"],
                         username=ios_xe1["username"],
                         password=ios_xe1["password"],
                         hostkey_verify=False) as m:

        # Get Configuration and State Info for Interface
        netconf_reply = m.get(netconf_filter)

        # Process the XML and store in useful dictionaries
        intf_details = xmltodict.parse(netconf_reply.xml)["rpc-reply"]["data"]
        intf_config = intf_details["interfaces"]["interface"]
        intf_info = intf_details["interfaces-state"]["interface"]

        print("")
        print("Interface Details:")
        print("  Name: {}".format(intf_config["name"]))
        print("  Description: {}".format(intf_config["description"]))
        print("  Type: {}".format(intf_config["type"]["#text"]))
        print("  MAC Address: {}".format(intf_info["phys-address"]))
        print("  Packets Input: {}".format(intf_info["statistics"]["in-unicast-pkts"]))
        print("  Packets Output: {}".format(intf_info["statistics"]["out-unicast-pkts"]))
```

BRKDEV-1368/netconf/example2.py

BRKDEV-1368/netconf/filter-ietf-interfaces.xml

Getting Interface Details with XML Filter

- example2.py: Retrieving info with ncclient
- Send <get> to retrieve config and state data
- Process and leverage XML within Python
- Report back current state of interface

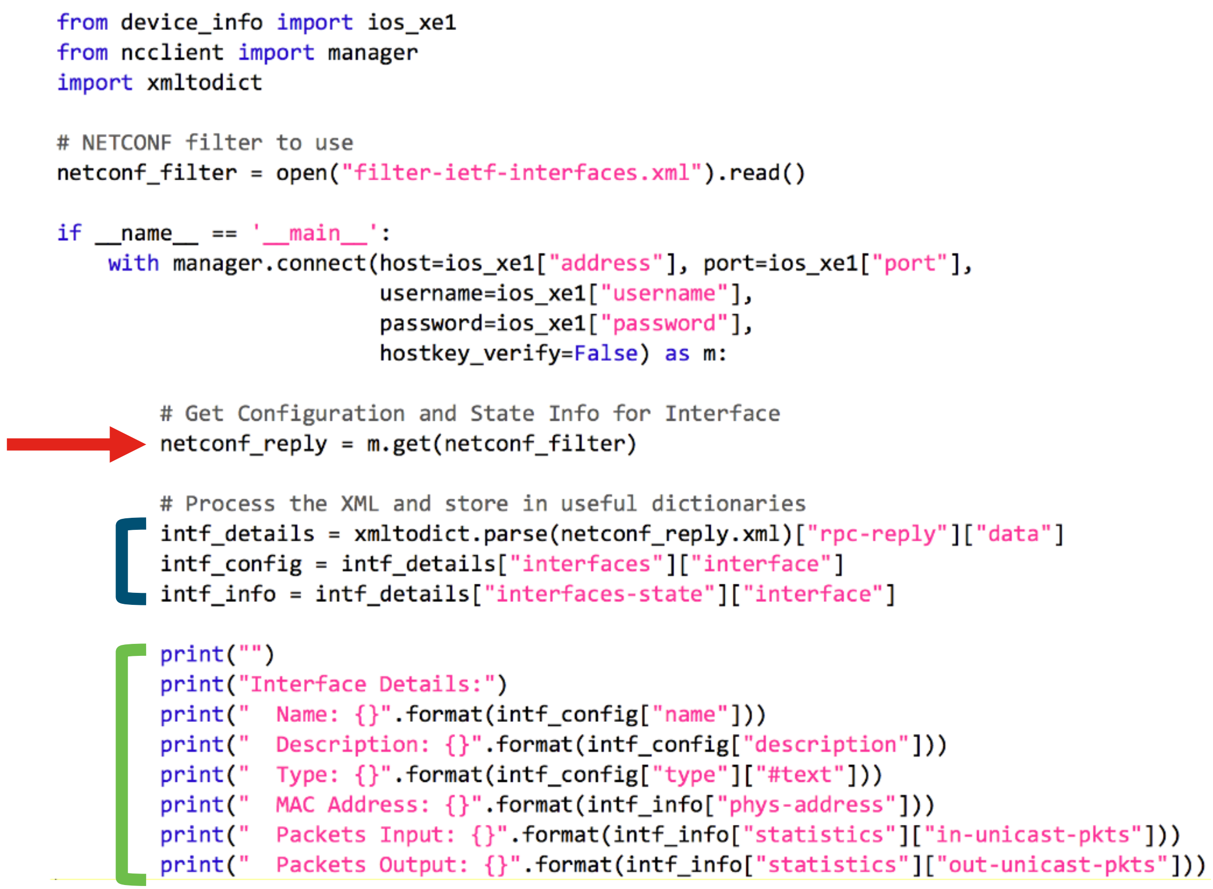


```
<filter>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
      <name>GigabitEthernet2</name>
    </interface>
  </interfaces>
  <interfaces-state xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
      <name>GigabitEthernet2</name>
    </interface>
  </interfaces-state>
</filter>
```

BRKDEV-1368/netconf/example2.py
BRKDEV-1368/netconf/filter-ietf-interfaces.xml

Getting Interface Details with XML Filter

- example2.py: Retrieving info with ncclient
- Send <get> to retrieve config and state data
- Process and leverage XML within Python
- Report back current state of interface



```
from device_info import ios_xe1
from ncclient import manager
import xmltodict

# NETCONF filter to use
netconf_filter = open("filter-ietf-interfaces.xml").read()

if __name__ == '__main__':
    with manager.connect(host=ios_xe1["address"], port=ios_xe1["port"],
                         username=ios_xe1["username"],
                         password=ios_xe1["password"],
                         hostkey_verify=False) as m:

        # Get Configuration and State Info for Interface
        netconf_reply = m.get(netconf_filter)

        # Process the XML and store in useful dictionaries
        intf_details = xmltodict.parse(netconf_reply.xml)["rpc-reply"]["data"]
        intf_config = intf_details["interfaces"]["interface"]
        intf_info = intf_details["interfaces-state"]["interface"]

        print("")
        print("Interface Details:")
        print("  Name: {}".format(intf_config["name"]))
        print("  Description: {}".format(intf_config["description"]))
        print("  Type: {}".format(intf_config["type"]["#text"]))
        print("  MAC Address: {}".format(intf_info["phys-address"]))
        print("  Packets Input: {}".format(intf_info["statistics"]["in-unicast-pkts"]))
        print("  Packets Output: {}".format(intf_info["statistics"]["out-unicast-pkts"]))
```

BRKDEV-1368/netconf/example2.py
BRKDEV-1368/netconf/filter-ietf-interfaces.xml

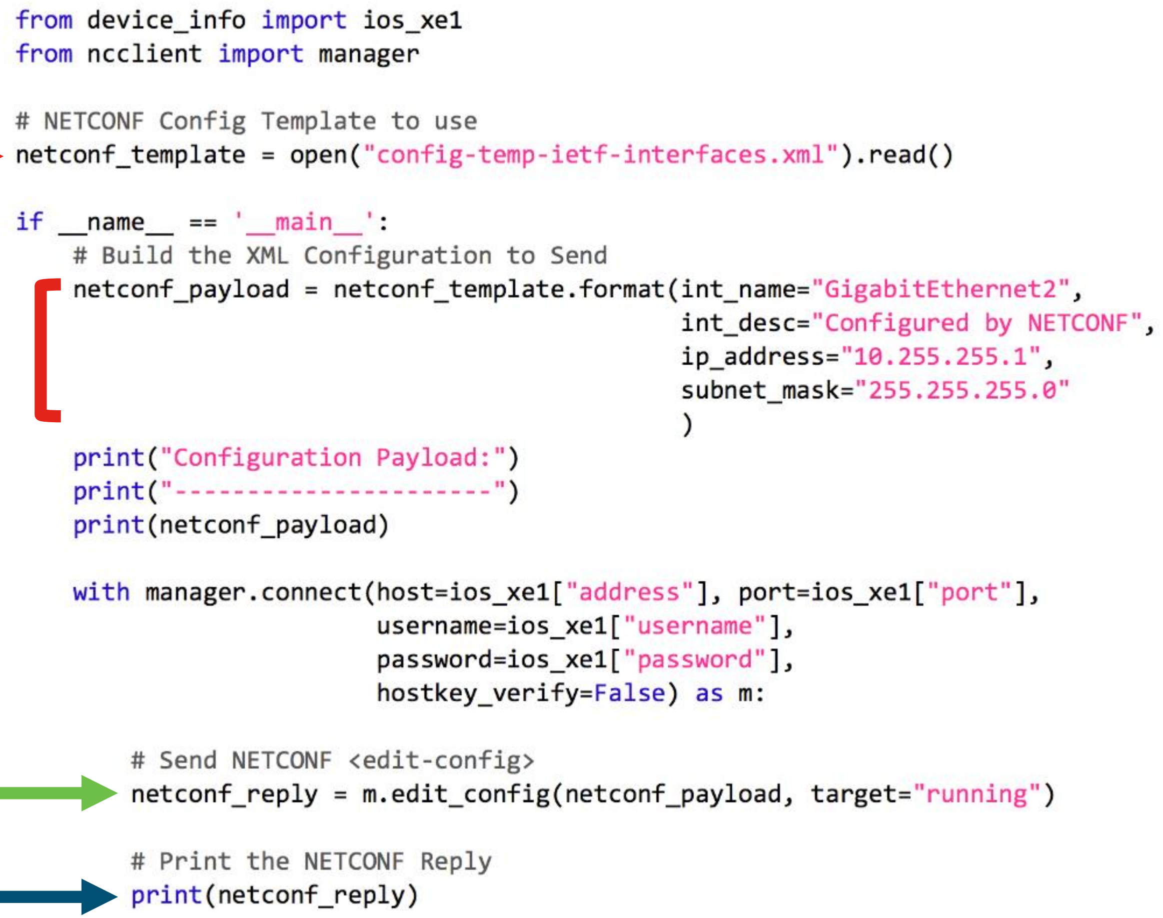
Getting Interface Details

```
DevNet$ python  example2.py

Interface Details:
  Name: GigabitEthernet1
  Description: DON'T TOUCH ME
  Type: ianaift:ethernetCsmacd
  MAC Address: 00:50:56:bb:74:d5
  Packets Input: 592268689
  Packets Output: 21839
```

BRKDEV-1368/netconf/example2.py
BRKDEV-1368/netconf/filter-ietf-interfaces.xml

Configuring Interface Details

- example3.py: Editing configuration with ncclient
- Constructing XML Config Payload for NETCONF
- Sending <edit-config> operation with ncclient
- Verify result



```python
from device_info import ios_xe1
from ncclient import manager

# NETCONF Config Template to use
netconf_template = open("config-temp-ietf-interfaces.xml").read()

if __name__ == '__main__':
    # Build the XML Configuration to Send
    netconf_payload = netconf_template.format(int_name="GigabitEthernet2",
                                              int_desc="Configured by NETCONF",
                                              ip_address="10.255.255.1",
                                              subnet_mask="255.255.255.0"
                                              )
    print("Configuration Payload:")
    print("----------------------")
    print(netconf_payload)

    with manager.connect(host=ios_xe1["address"], port=ios_xe1["port"],
                         username=ios_xe1["username"],
                         password=ios_xe1["password"],
                         hostkey_verify=False) as m:

        # Send NETCONF <edit-config>
        netconf_reply = m.edit_config(netconf_payload, target="running")

        # Print the NETCONF Reply
        print(netconf_reply)
```

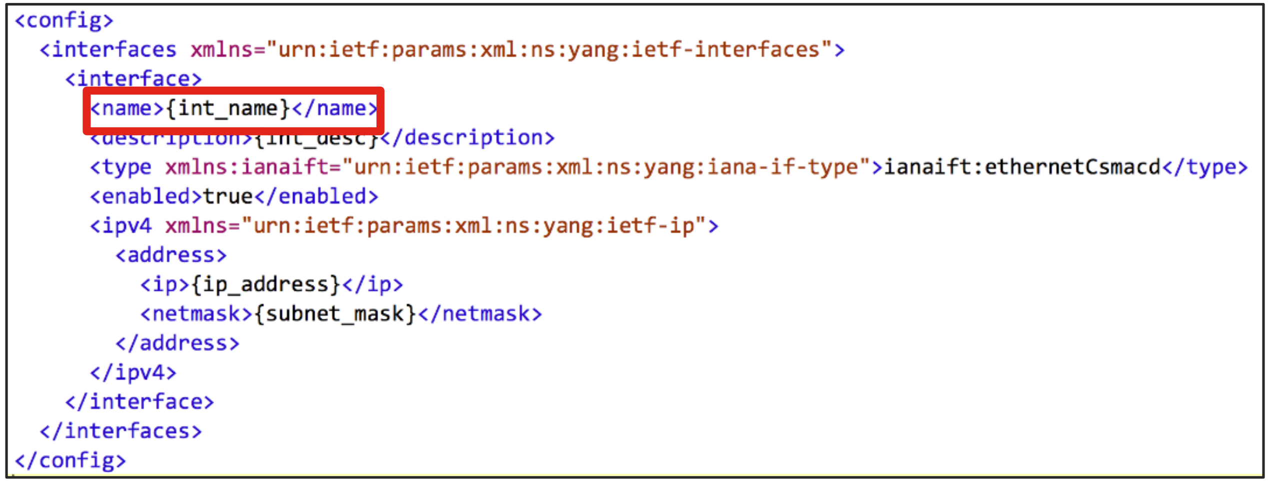
BRKDEV-1368/netconf/config-temp-ietf-interfaces.xml

BRKDEV-1368/netconf/example3.py

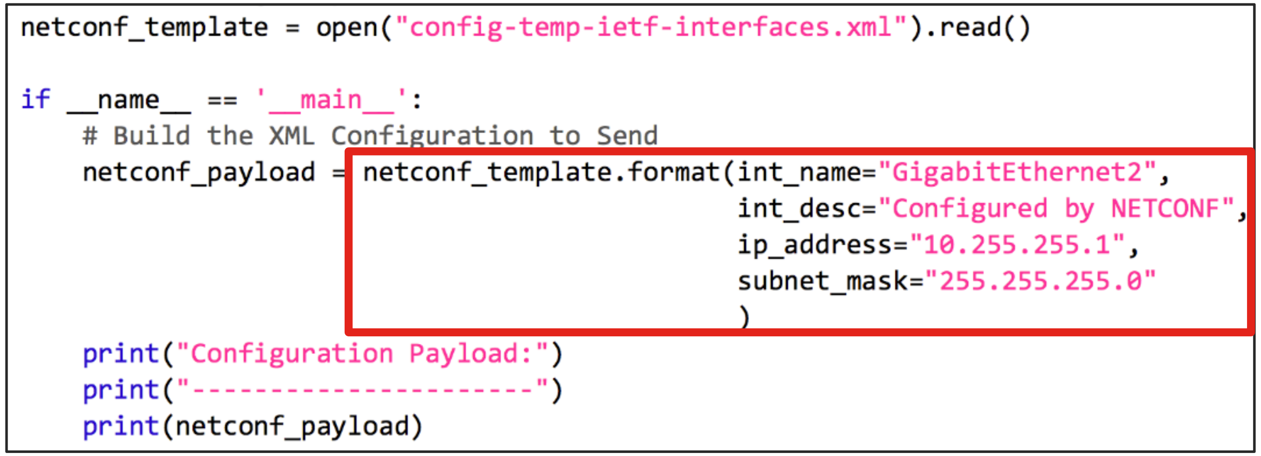
Configuring Interface Details

- example3.py: Editing configuration with ncclient
- Constructing XML Config Payload for NETCONF
- Sending <edit-config> operation with ncclient
- Verify result

config-temp-ietf-interfaces.xml



```
<config>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
    <interface>
      <name>{int_name}</name>
      <description>{int_desc}</description>
      <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">ianaift:ethernetCsmacd</type>
      <enabled>true</enabled>
      <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
        <address>
          <ip>{ip_address}</ip>
          <netmask>{subnet_mask}</netmask>
        </address>
      </ipv4>
    </interface>
  </interfaces>
</config>
```

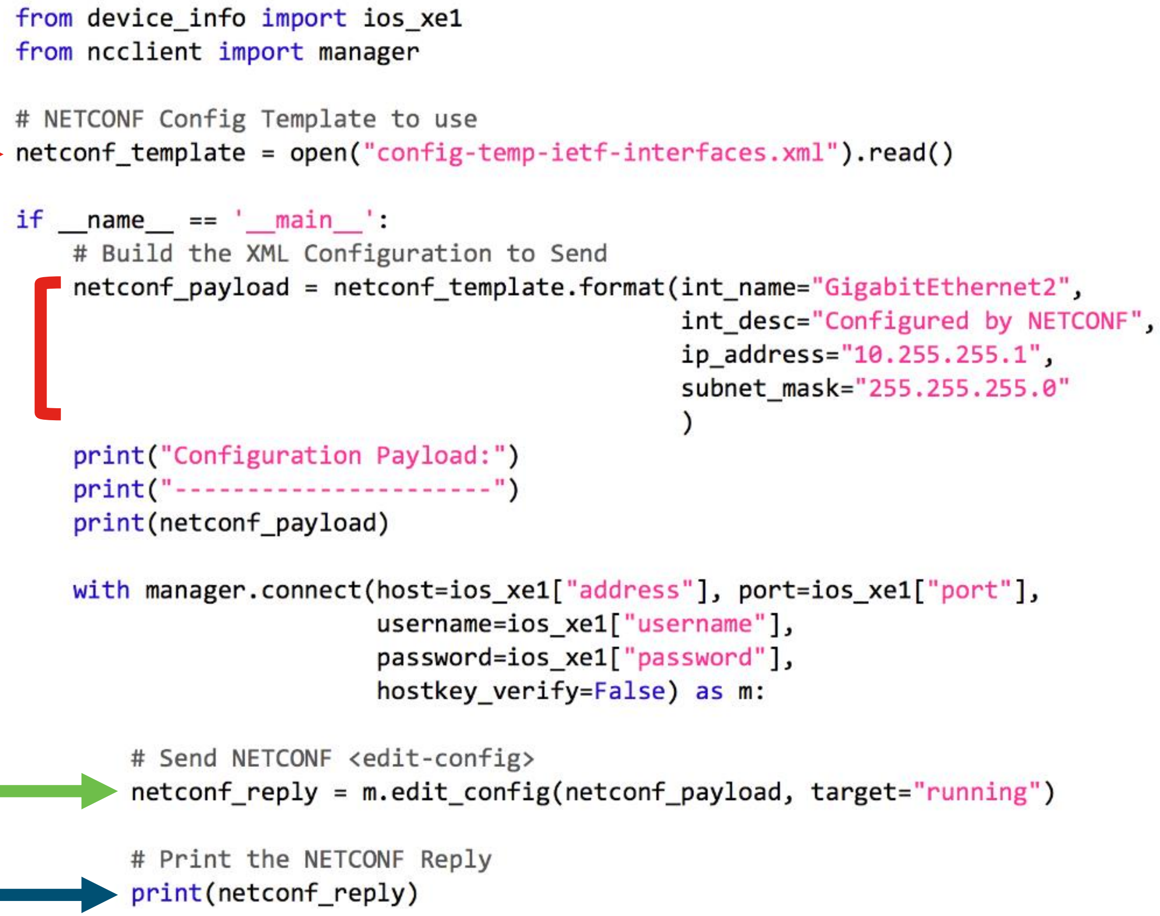


```
netconf_template = open("config-temp-ietf-interfaces.xml").read()

if __name__ == '__main__':
    # Build the XML Configuration to Send
    netconf_payload = netconf_template.format(int_name="GigabitEthernet2",
                                              int_desc="Configured by NETCONF",
                                              ip_address="10.255.255.1",
                                              subnet_mask="255.255.255.0"
                                              )
    print("Configuration Payload:")
    print("----------------------")
    print(netconf_payload)
```

BRKDEV-1368/netconf/config-temp-ietf-interfaces.xml
BRKDEV-1368/netconf/example3.py

Configuring Interface Details

- example3.py: Editing configuration with ncclient
- Constructing XML Config Payload for NETCONF
- Sending <edit-config> operation with ncclient
- Verify result



```python
from device_info import ios_xe1
from ncclient import manager

# NETCONF Config Template to use
netconf_template = open("config-temp-ietf-interfaces.xml").read()

if __name__ == '__main__':
    # Build the XML Configuration to Send
    netconf_payload = netconf_template.format(int_name="GigabitEthernet2",
                                              int_desc="Configured by NETCONF",
                                              ip_address="10.255.255.1",
                                              subnet_mask="255.255.255.0"
                                              )

    print("Configuration Payload:")
    print("----------------------")
    print(netconf_payload)

    with manager.connect(host=ios_xe1["address"], port=ios_xe1["port"],
                         username=ios_xe1["username"],
                         password=ios_xe1["password"],
                         hostkey_verify=False) as m:

        # Send NETCONF <edit-config>
        netconf_reply = m.edit_config(netconf_payload, target="running")

        # Print the NETCONF Reply
        print(netconf_reply)
```

BRKDEV-1368/netconf/config-temp-ietf-interfaces.xml
BRKDEV-1368/netconf/example3.py

Configuring Interface Details

```
DevNet$ python -i example3.py
Configuration Payload:
----------------------
<config>
  <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
          <name>GigabitEthernet2</name>
          <description>Configured by NETCONF</description>
          <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">
                   ianaift:ethernetCsmacd
           </type>
          <enabled>true</enabled>
          <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">
                <address>
                  <ip>10.255.255.1</ip>
                  <netmask>255.255.255.0</netmask>
                </address>
          </ipv4>
        </interface>
  </interfaces>
</config>

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns=”urn.." message-id=”..9784" xmlns:nc="urn..">
  <ok/>
</rpc-reply>
```

Example edited for simplicity and brevity

BRKDEV-1368/netconf/config-temp-ietf-interfaces.xml
BRKDEV-1368/netconf/example3.py

NETCONF Summary

NETCONF Summary

- The elements of the NETCONF transport protocol
- How to leverage ncclient to use NETCONF in Python
- Examples retrieving and configuring data from a NETCONF Agent

Understanding RESTCONF

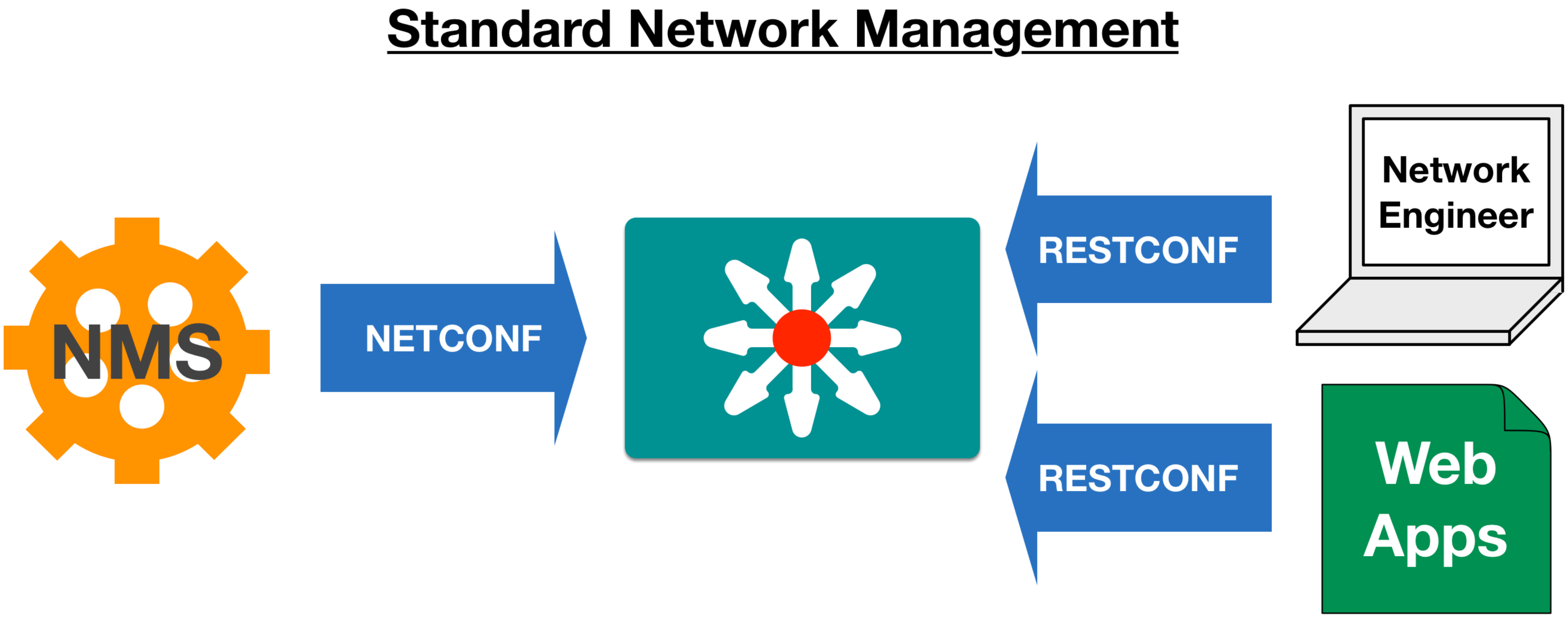
RESTCONF Details

“an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG…”

https://tools.ietf.org/html/rfc8040

- RFC 8040 - January 2017
- Uses HTTPS for transport
- Tightly coupled to the YANG data model definitions
- Provides JSON or XML data formats

What about NETCONF?



RESTCONF Protocol Stack & Transport

RESTCONF Protocol Stack

Content	Configuration / Operational Data	`XML or JSON`
Operations	Actions to Take	`GET, POST, PUT, PATCH, DELETE`
Transport	TCP/IP Method	HTTPS

Operations – HTTP CRUD

RESTCONF	NETCONF
GET	<get> , <get-config>
POST	<edit-config> (operation=" create")
PUT	<edit-config> (operation=" create/replace")
PATCH	<edit-config> (operation=" merge")
DELETE	<edit-config> (operation=" delete")

Content - XML or JSON

HTTP Headers

- **Content-Type**: Specify the type of data being sent from the client
- **Accept**: Specify the type of data being requested by the client

RESTCONF MIME Types

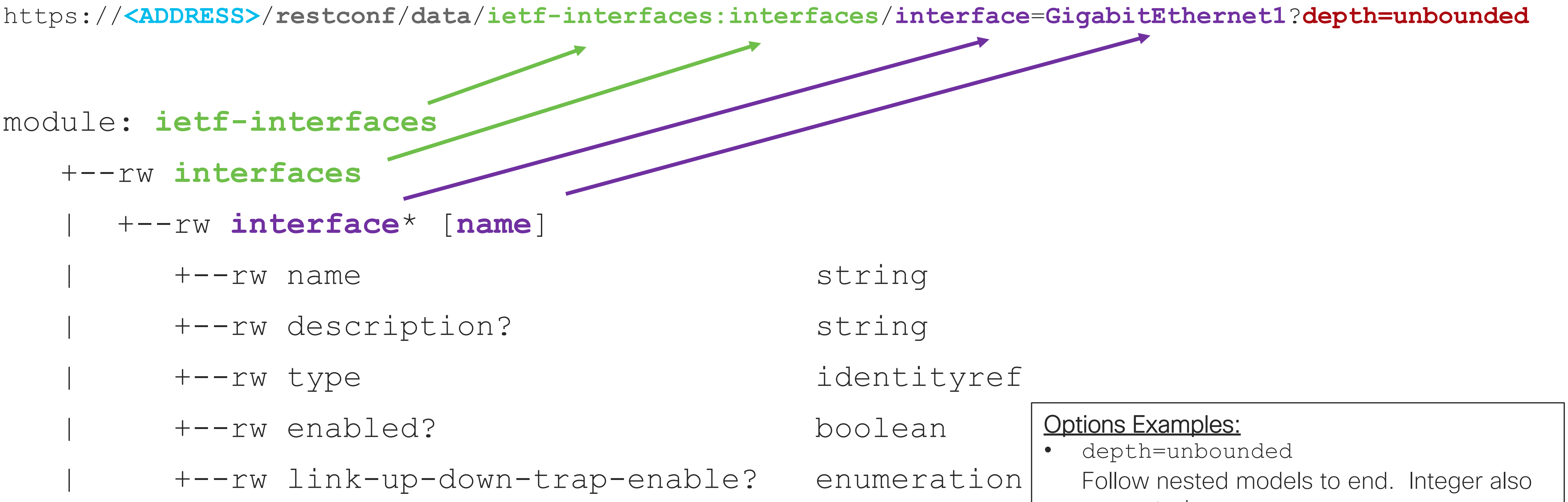
- application/yang-data+json
- application/yang-data+xml

Constructing RESTCONF URIs for Data Resources

```
https://<ADDRESS>/<ROOT>/data/<[YANG MODULE:]CONTAINER>/<LEAF>[?<OPTIONS>]
```

- ADDRESS - Of the RESTCONF Agent
- ROOT - The main entry point for RESTCONF requests.
  Discoverable at `https://<ADDRESS>/.well-known/host-meta`
- data - The RESTCONF API resource type for data
  - *The "operations" resource type used to access RPC operations available*
- [YANG MODULE:]CONTAINER - The base model container being used. Providing the module name is optional.
- LEAF - An individual element from within the container
- [?<OPTIONS>] - optional parameters that impact returned results.

URL Creation Review



```
https://<ADDRESS>/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet1?depth=unbounded
```

```
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
```

Options Examples:

- `depth=unbounded`
  Follow nested models to end. Integer also supported
- `content=[`**`all`**`, config, nonconfig]`
  Query option controls type of data returned.
- `fields=`***`expr`***
  Limit what leafs are returned

Key:
https://<ADDRESS>/<ROOT>/data>/<[YANG MODULE:]CONTAINER>/<LEAF>[?<OPTIONS>]

Using RESTCONF with Postman

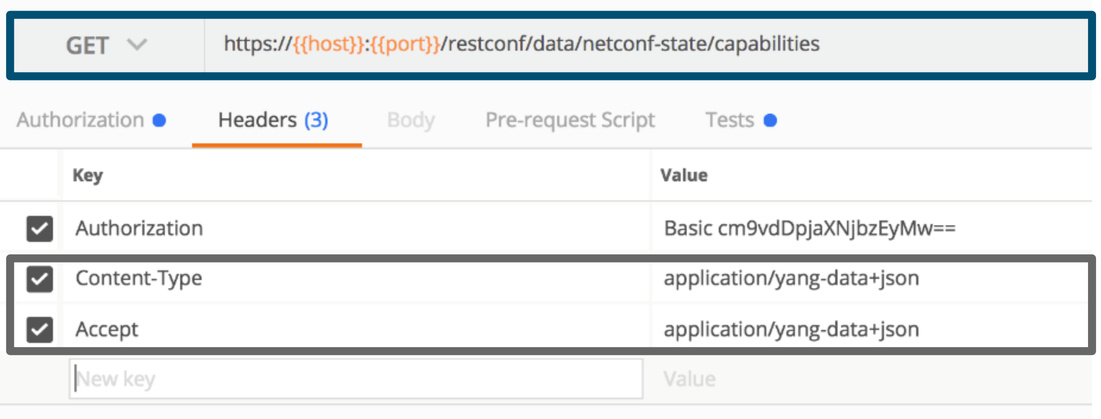
Postman: Powerful but Simple REST API Client

- Quickly test APIs in GUI
- Save APIs into Collections for reuse
- Manage multiple environments
- Auto generate code from API calls
- Standalone Application or Chrome Plugin

https://www.getpostman.com

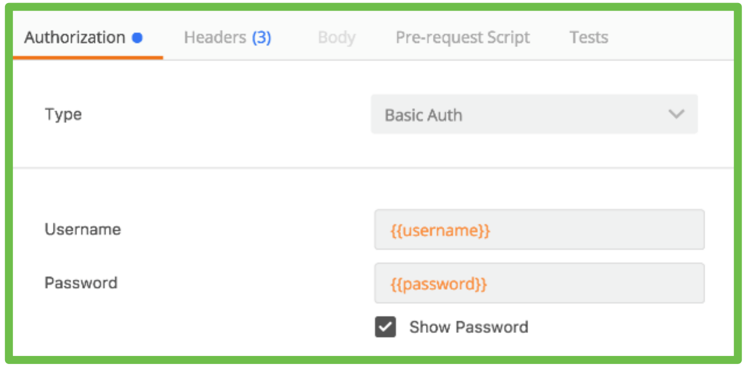
Step 1: Get Capabilities List via RESTCONF

- GET `/restconf/data/netconf-state/capabilities`
- Add RESTCONF Headers
  - Content-Type and Accept `application/yang-data+json` (or xml)
- Configure Basic Auth with username and password variables



https://{{host}}:{{port}}/restconf/data/netconf-state/capabilities

	Key	Value
☑	Authorization	Basic cm9vdDpjaXNjbzEyMw==
☑	Content-Type	application/yang-data+json
☑	Accept	application/yang-data+json
	New key	Value



Type Basic Auth

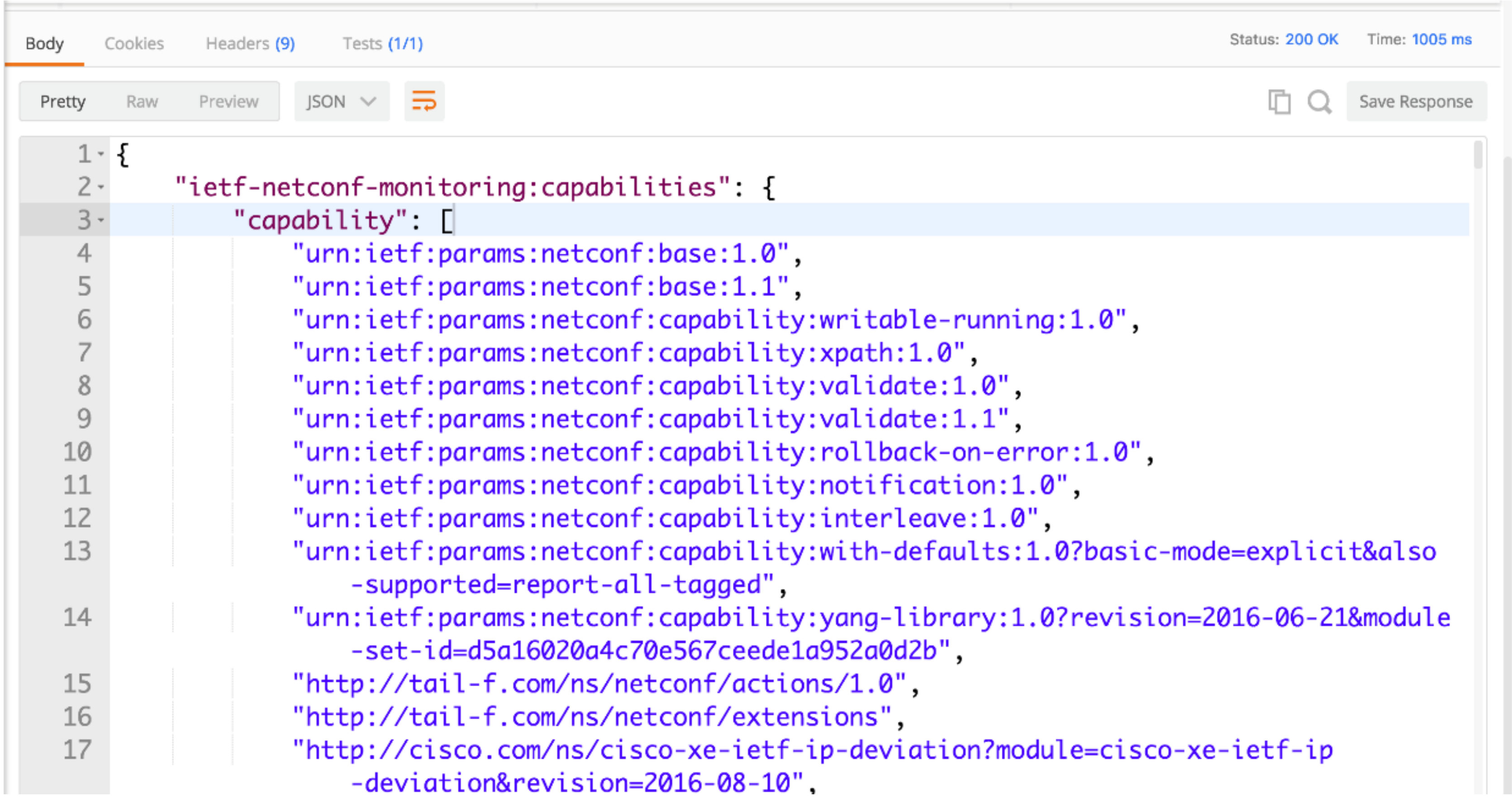
Username {{username}}

Password {{password}}

☑ Show Password

Step 1: Get Capabilities List via RESTCONF

- Send and review results

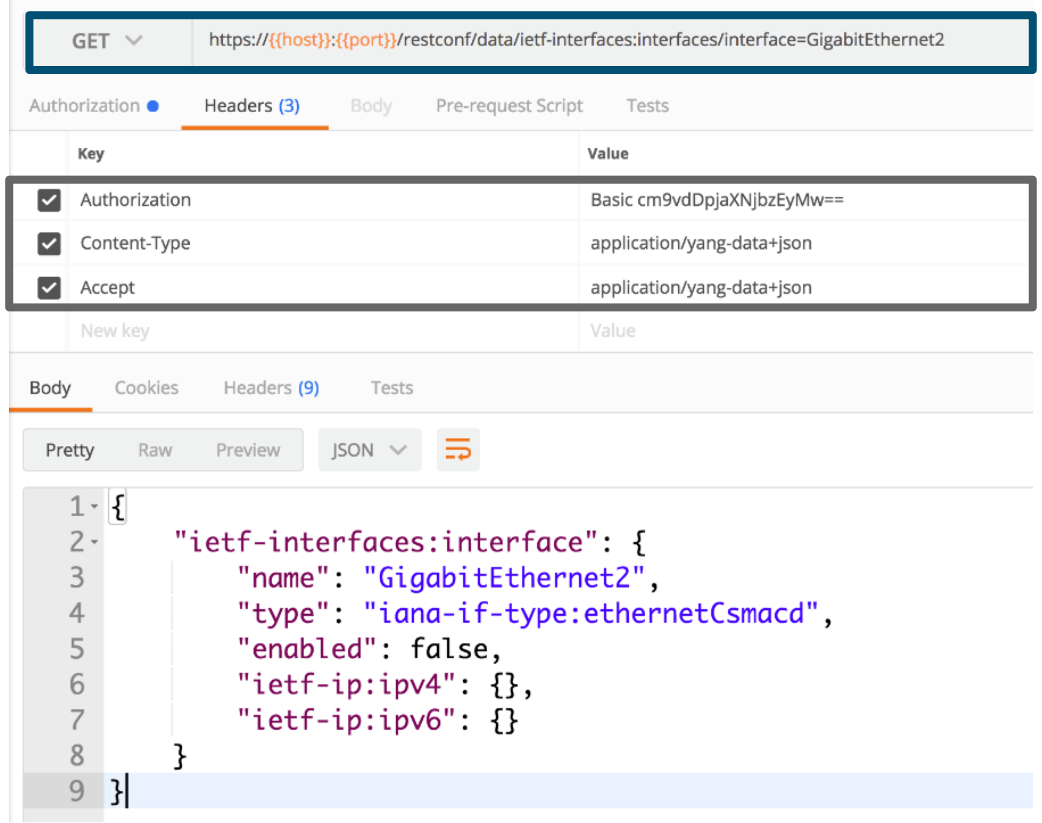


```
{
    "ietf-netconf-monitoring:capabilities": {
        "capability": [
            "urn:ietf:params:netconf:base:1.0",
            "urn:ietf:params:netconf:base:1.1",
            "urn:ietf:params:netconf:capability:writable-running:1.0",
            "urn:ietf:params:netconf:capability:xpath:1.0",
            "urn:ietf:params:netconf:capability:validate:1.0",
            "urn:ietf:params:netconf:capability:validate:1.1",
            "urn:ietf:params:netconf:capability:rollback-on-error:1.0",
            "urn:ietf:params:netconf:capability:notification:1.0",
            "urn:ietf:params:netconf:capability:interleave:1.0",
            "urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&also-supported=report-all-tagged",
            "urn:ietf:params:netconf:capability:yang-library:1.0?revision=2016-06-21&module-set-id=d5a16020a4c70e567ceede1a952a0d2b",
            "http://tail-f.com/ns/netconf/actions/1.0",
            "http://tail-f.com/ns/netconf/extensions",
            "http://cisco.com/ns/cisco-xe-ietf-ip-deviation?module=cisco-xe-ietf-ip-deviation&revision=2016-08-10",
```

Automate Your Network with RESTCONF

Getting Interface Details

- GET `restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet2`
- Configure Auth and Headers



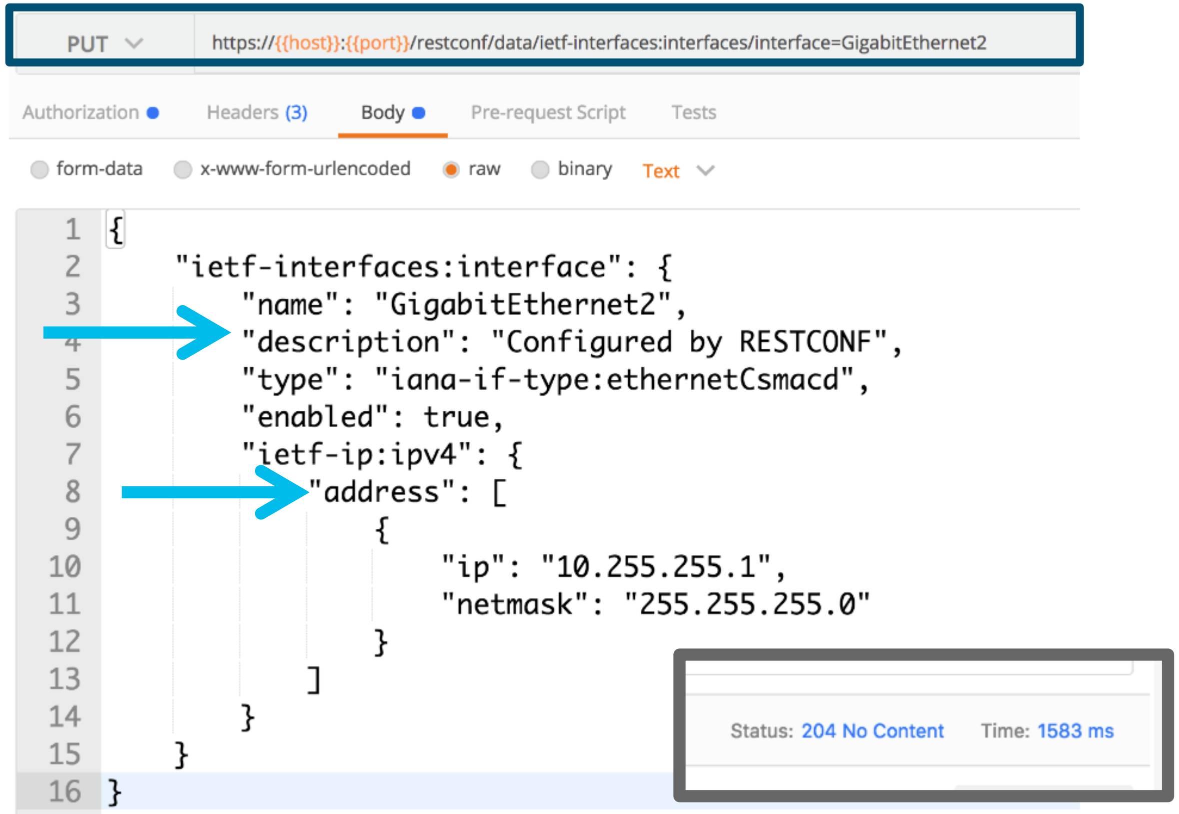
GET https://{{host}}:{{port}}/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet2

	Key	Value
☑	Authorization	Basic cm9vdDpjaXNjbzEyMw==
☑	Content-Type	application/yang-data+json
☑	Accept	application/yang-data+json
	New key	Value

```
{
    "ietf-interfaces:interface": {
        "name": "GigabitEthernet2",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false,
        "ietf-ip:ipv4": {},
        "ietf-ip:ipv6": {}
    }
}
```

Configuring Interface Details

- PUT `restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet2`
- Configure Auth and Headers
- Configure Body (raw)
- Send and check status code



PUT https://{{host}}:{{port}}/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet2

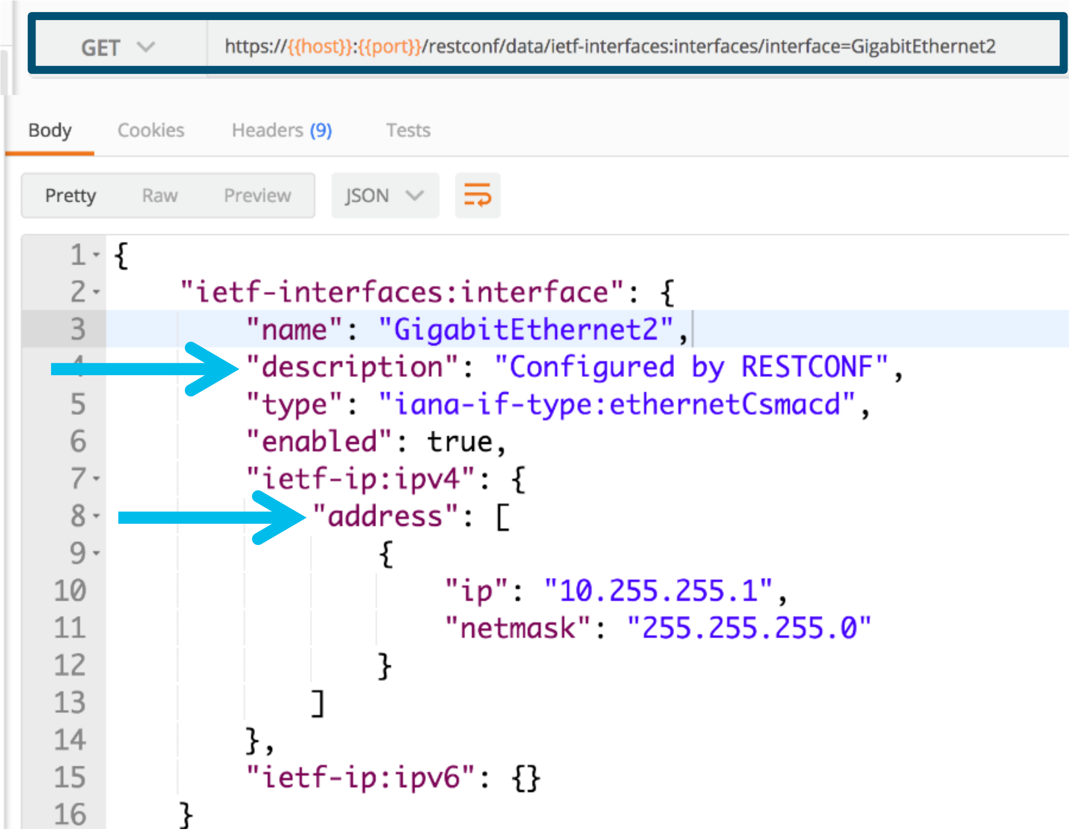
○ form-data ○ x-www-form-urlencoded ● raw ○ binary Text

```
{
    "ietf-interfaces:interface": {
        "name": "GigabitEthernet2",
        "description": "Configured by RESTCONF",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-ip:ipv4": {
            "address": [
                {
                    "ip": "10.255.255.1",
                    "netmask": "255.255.255.0"
                }
            ]
        }
    }
}
```

Status: 204 No Content   Time: 1583 ms

Configuring Interface Details - Verification

- GET
  `restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet2`
- Configure Auth and Headers
- Check that the new config was successful



GET https://{{host}}:{{port}}/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet2

```
{
    "ietf-interfaces:interface": {
        "name": "GigabitEthernet2",
        "description": "Configured by RESTCONF",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": true,
        "ietf-ip:ipv4": {
            "address": [
                {
                    "ip": "10.255.255.1",
                    "netmask": "255.255.255.0"
                }
            ]
        },
        "ietf-ip:ipv6": {}
    }
```

RESTCONF Summary

Review

- The elements of the RESTCONF transport protocol
- How to leverage Postman to use RESTCONF
- Examples retrieving and configuring data using RESTCONF

Questions?

Review

- The Road to Model Driven Programmability
- Introduction to YANG Data Models
- Introduction to NETCONF
- Introduction to RESTCONF
- Conclusion and Q/A

Note: All code samples referenced in this presentation are available at
https://github.com/CiscoDevNet/BRKDEV-1368

What do do next?

- Resources
  - Overview of the 2002 IAB Network Management Workshop
  - Network Configuration Protocol (NETCONF)
  - The YANG 1.1 Data Modeling Language
  - RESTCONF Protocol
  - YANG Development Kit (YDK)

- DevNet Learning Labs
  - Introduction to Device Level Interfaces – NETCONF/YANG
  - NETCONF/YANG on Nexus
  - Home Lab: Using NETCONF/YANG from your Desktop OS
- Blogs and Videos
  - Using CLI as Training Wheels with NETCONF/YANG
  - Simplifying Network Programmability with Model Driven APIs
  - Network Device APIs Video Lessons

Got more questions? Stay in touch!

Hank Preston

- hapresto@cisco.com
- @hfpreston
- http://github.com/hpreston

CISCO

DEVNET

LEARN CODE INSPIRE CONNECT

developer.cisco.com

- @CiscoDevNet
- facebook.com/ciscodevnet/
- http://github.com/CiscoDevNet

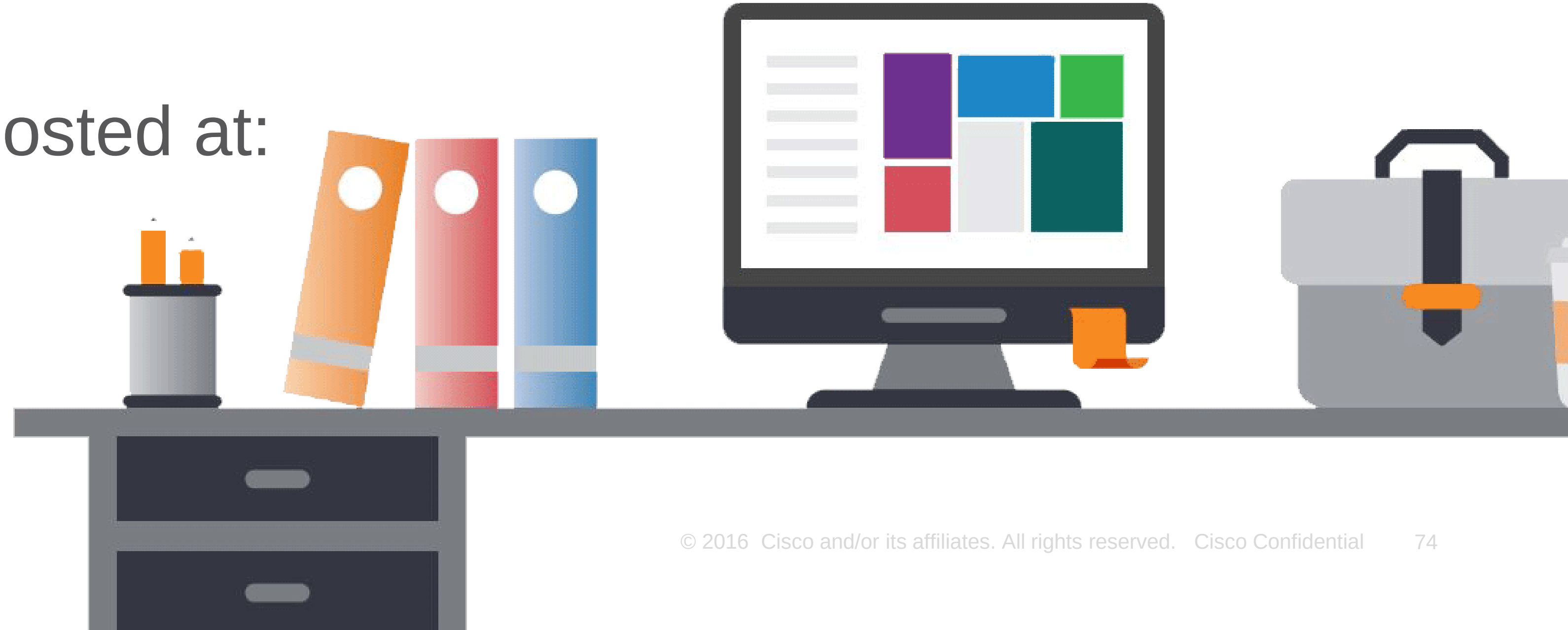
Remaining Sessions:

- #8 Automating Spark with Cloud Integration
- #9 Using Python to Automate Spark
- #10 Making Spark Interactive with ChatOps & ChatBots

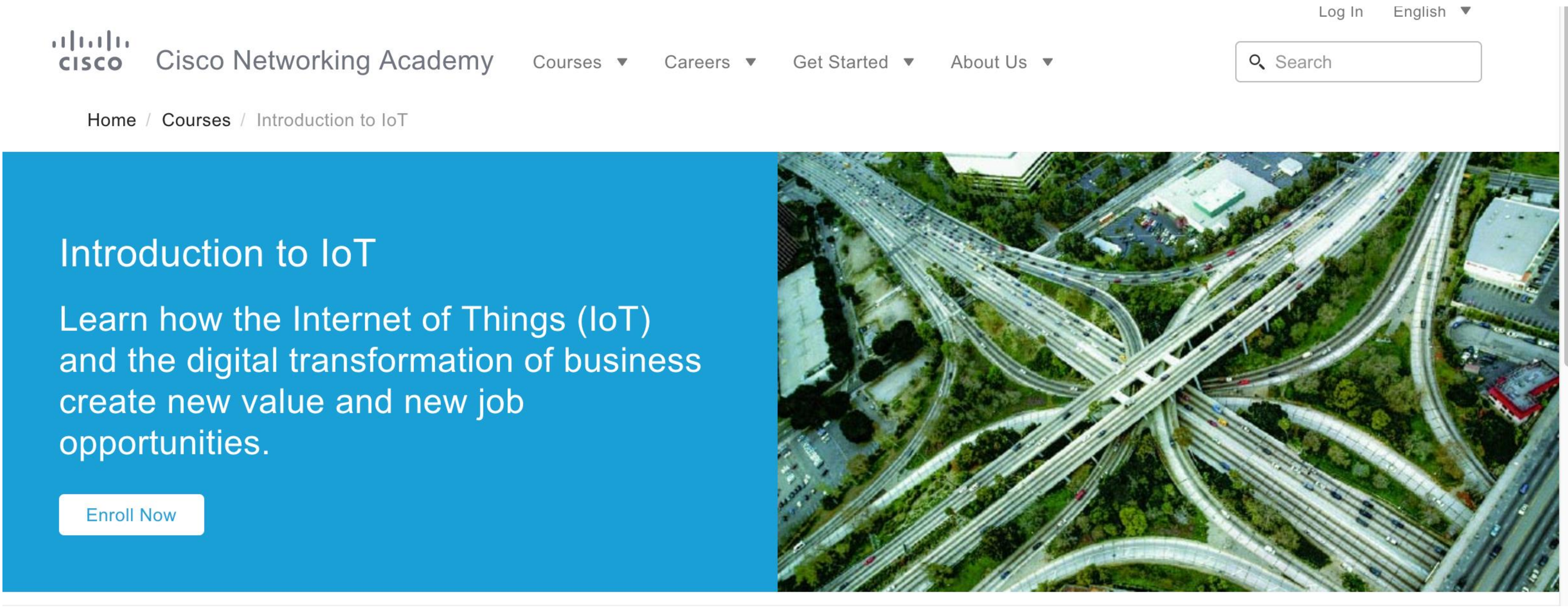
📅 - TBD

Registration will soon be posted at:

http://bit.ly/devnetseries



This is the Digital Transformation



Introduction to IoT

Learn how the Internet of Things (IoT) and the digital transformation of business create new value and new job opportunities.

Self-enroll today to learn more @ http://bit.ly/introiot cisco