Program Network Devices using their APIs

A Look at Model Driven Programmability with RESTCONF and NETCONF

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

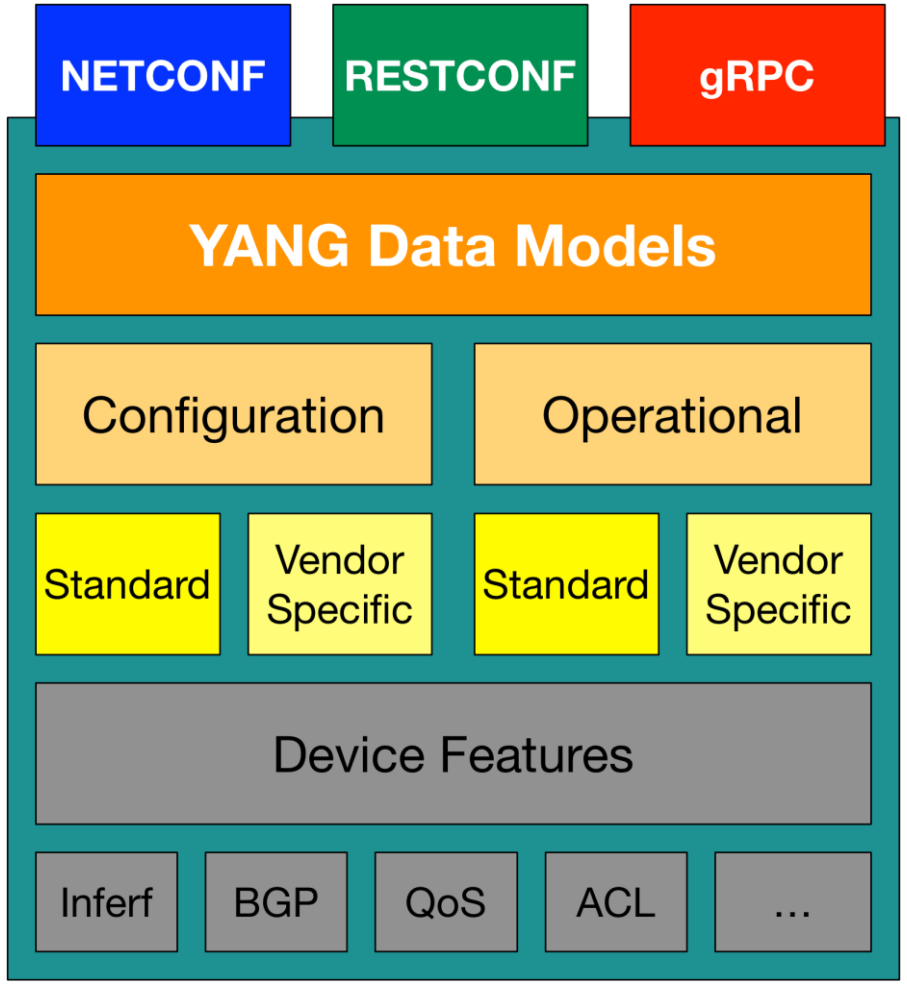
Agenda

- What is Model Driven Programmability
- A Word about YANG
- A Look at RESTCONF
- A Look at NETCONF

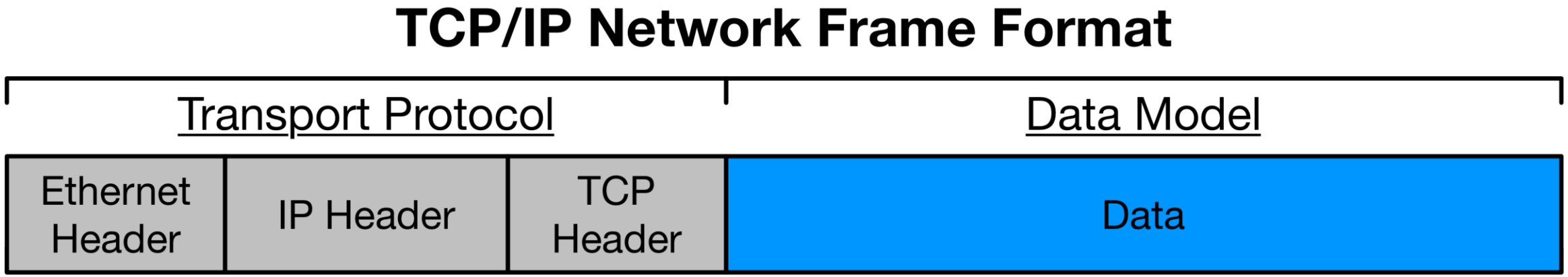
What is Model Driven Programmability

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)



- NETCONF
- RESTCONF
- gRPC

- YANG

A Word about 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";
      }
    }
}
```

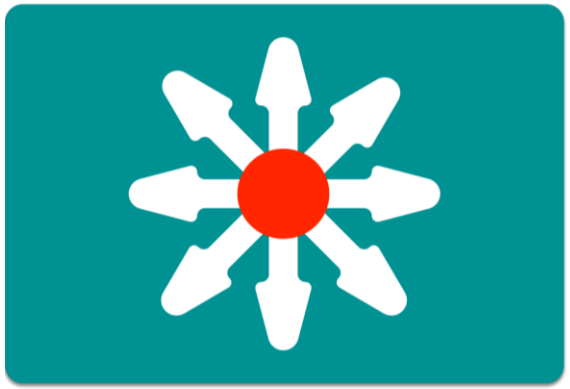
Example edited for simplicity and brevity

What is a Data Model?

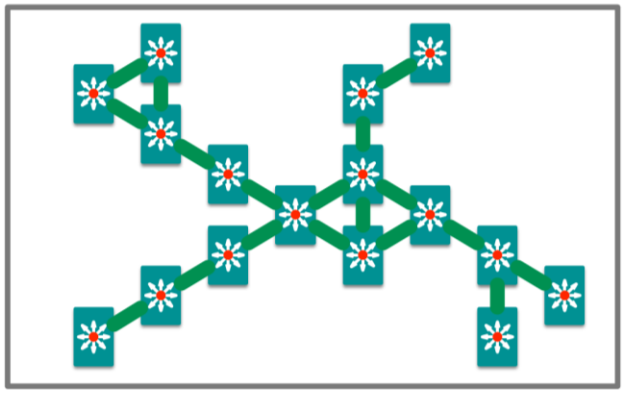
A data model is simply a well understood and agreed upon method to describe "something". As an example, consider this simple "data model" for a person.

- *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

Where do Models Come From?

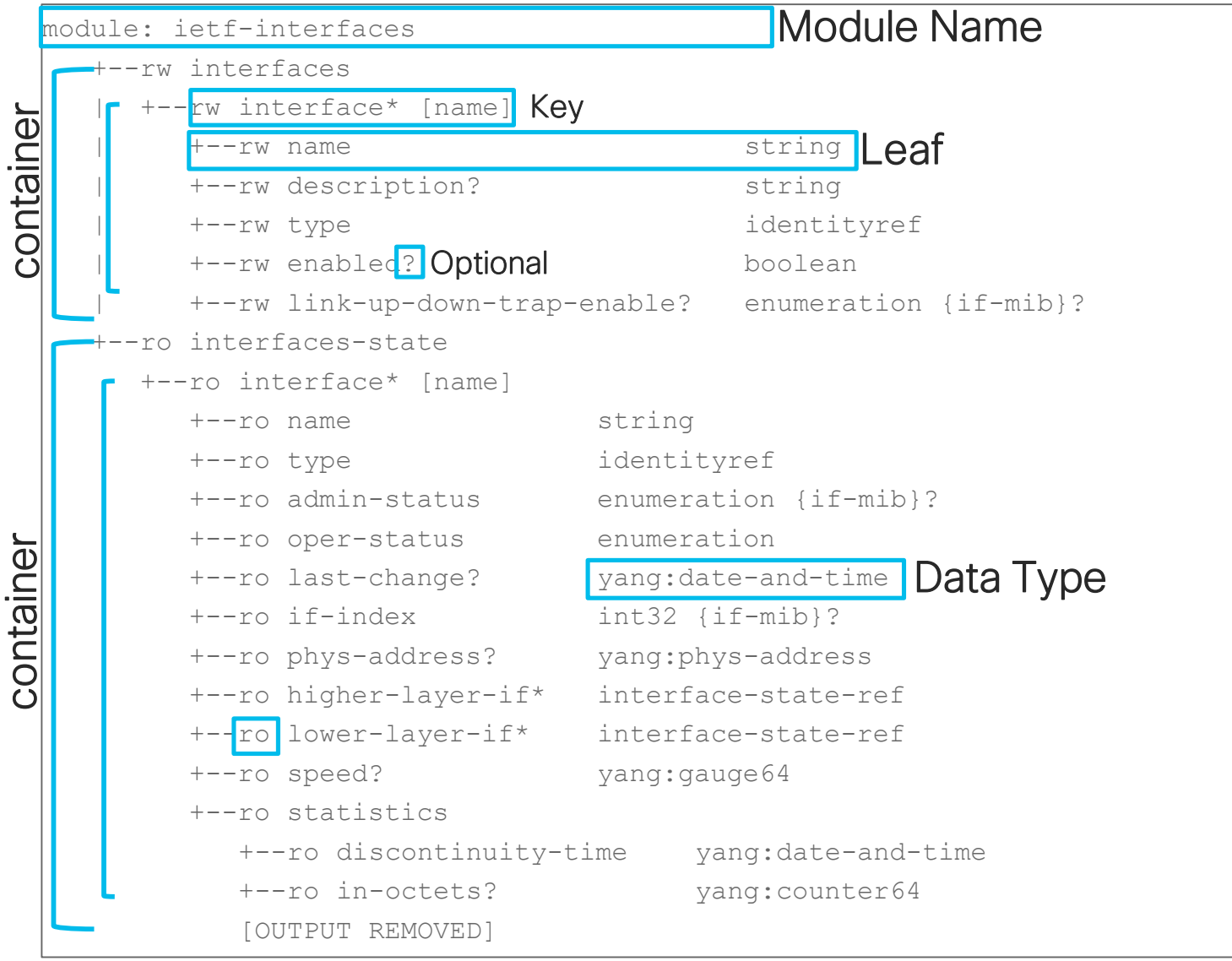
- **Standard definition**
  (IETF, ITU, OpenConfig, etc.)
- Compliant with standard
  `ietf-diffserv-policy.yang`
  `ietf-diffserv-classifer.yang`
  `ietf-diffserv-target.yang`

- **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

Using pyang

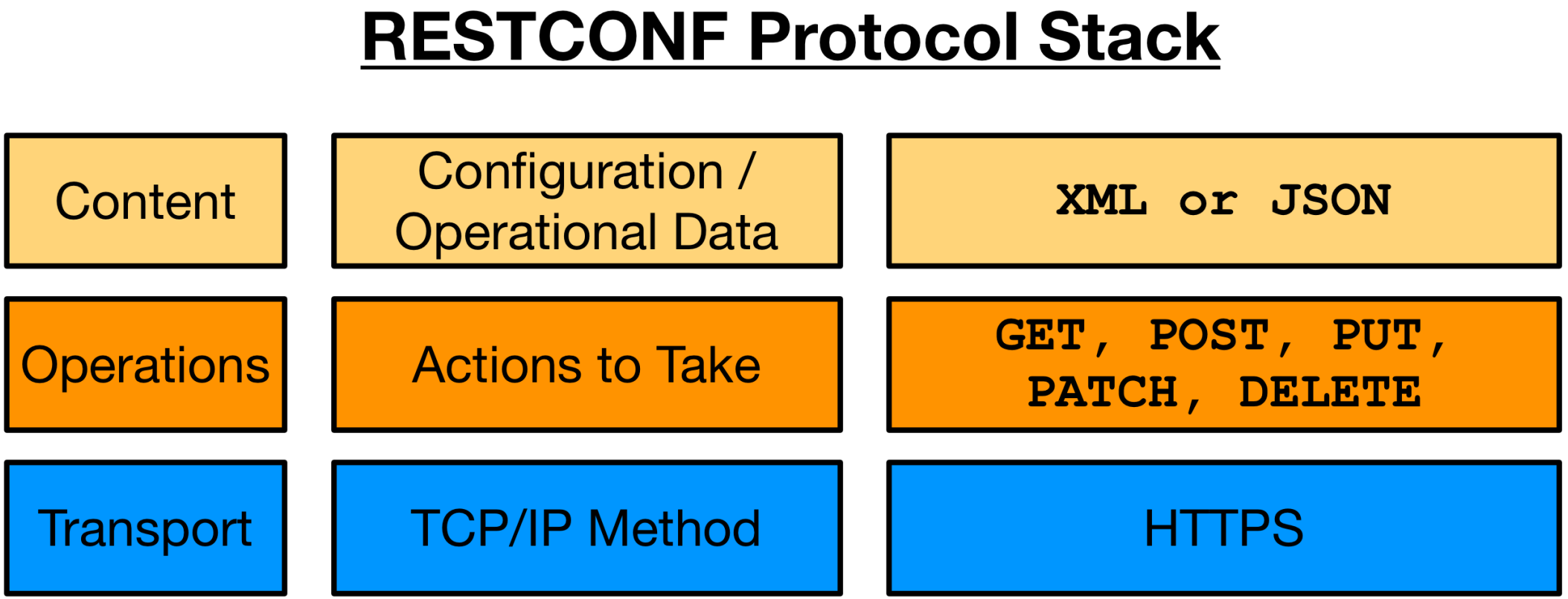
- Python YANG Library
- Validate and display YANG files
- Many formats for display
  - Text: tree
  - HTML: jstree



Example edited for simplicity and brevity

A Look at RESTCONF

RESTCONF Protocol Stack & Transport



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

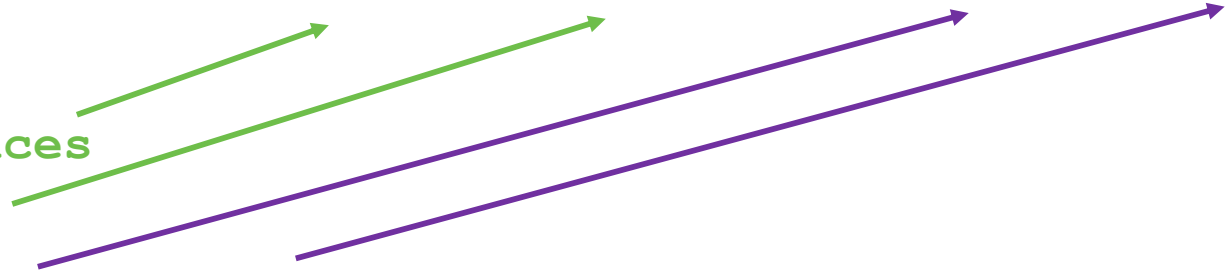
Constructing RESTCONF URIs for Data Resources

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:

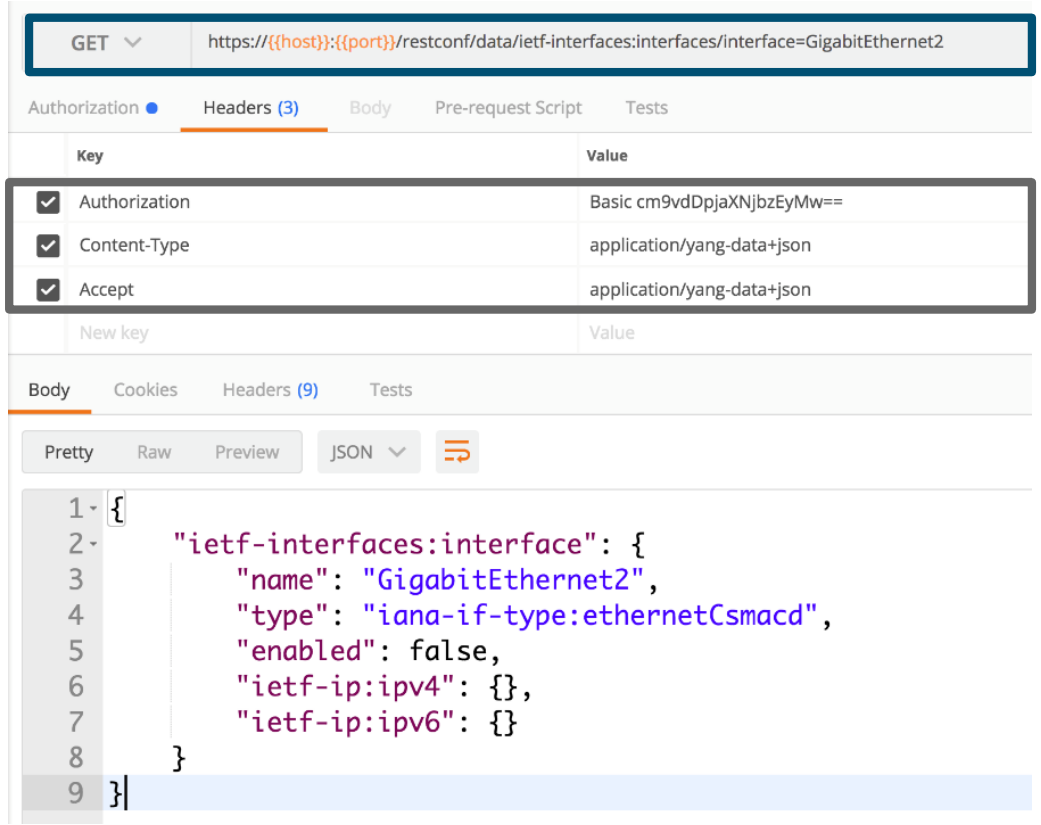
- `depth=unbounded`
  Follow nested models to end. Integer also supported
- `content=[all, config, nonconfig]`
  Query option controls type of data returned.
- `fields=expr`
  Limit what leafs are returned

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

RESTCONF in Action

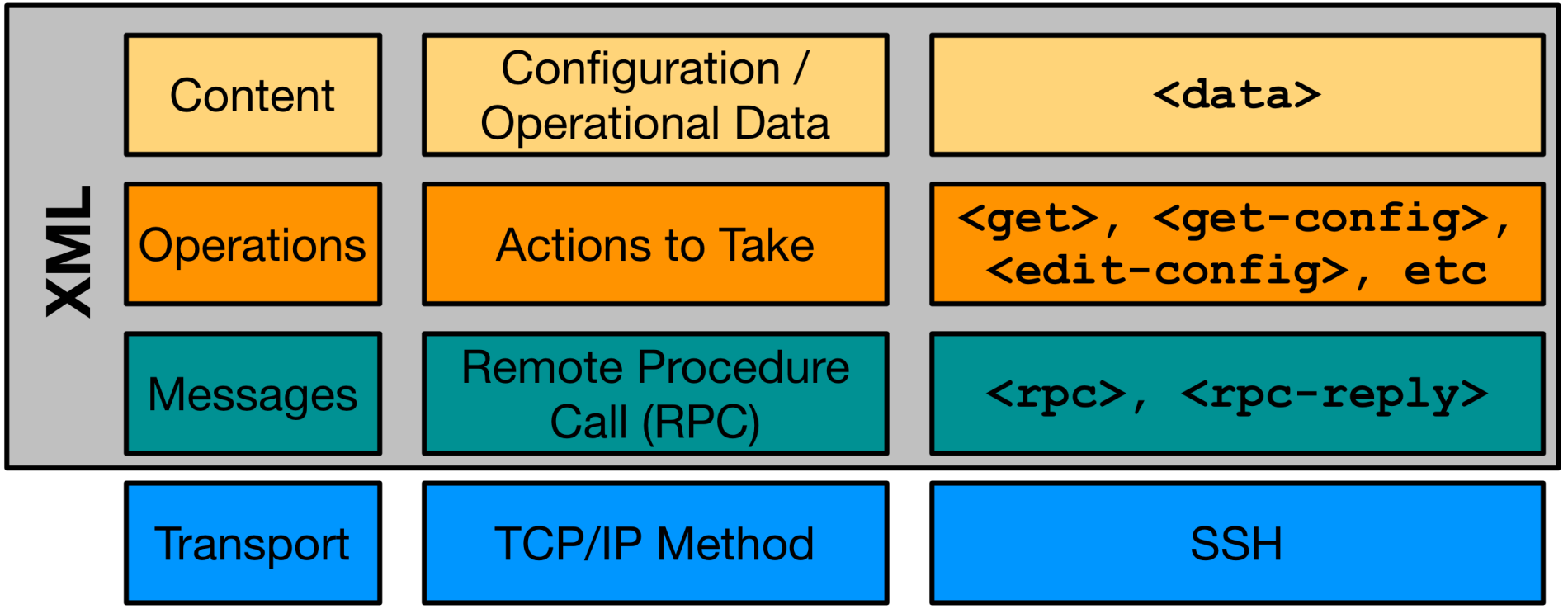
Getting Interface Details

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



A Look at NETCONF

NETCONF Protocol Stack



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 in Action

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>]]>]]>
```

Server (Agent) sends hello

Don’t NETCONF Like this!

```
<?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

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

```
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: GigabitEthernet2
  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

Questions?

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)
- Code Samples

- 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

DEVNET

developer.cisco.com