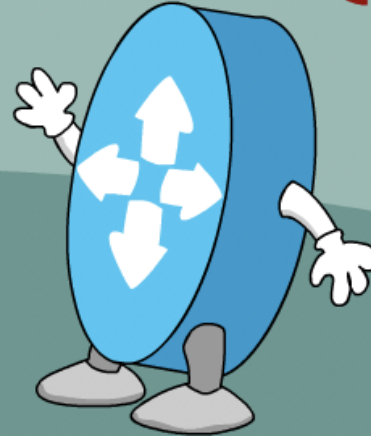




# NETDEVOPS {LIVE!}



DEVNET

## Deep Dive Into Model Driven Programmability with NETCONF and YANG

Bryan Byrne, CCIE 25607 R/S

Technical Solutions

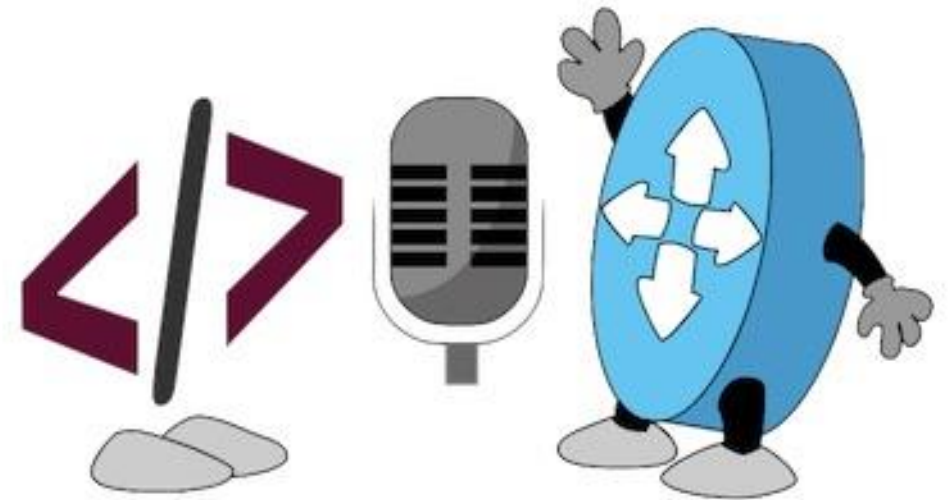
Twitter: @bryan25607

Season 1, Talk 3

<https://developer.cisco.com/netdevops/live>

# What are we going to talk about?

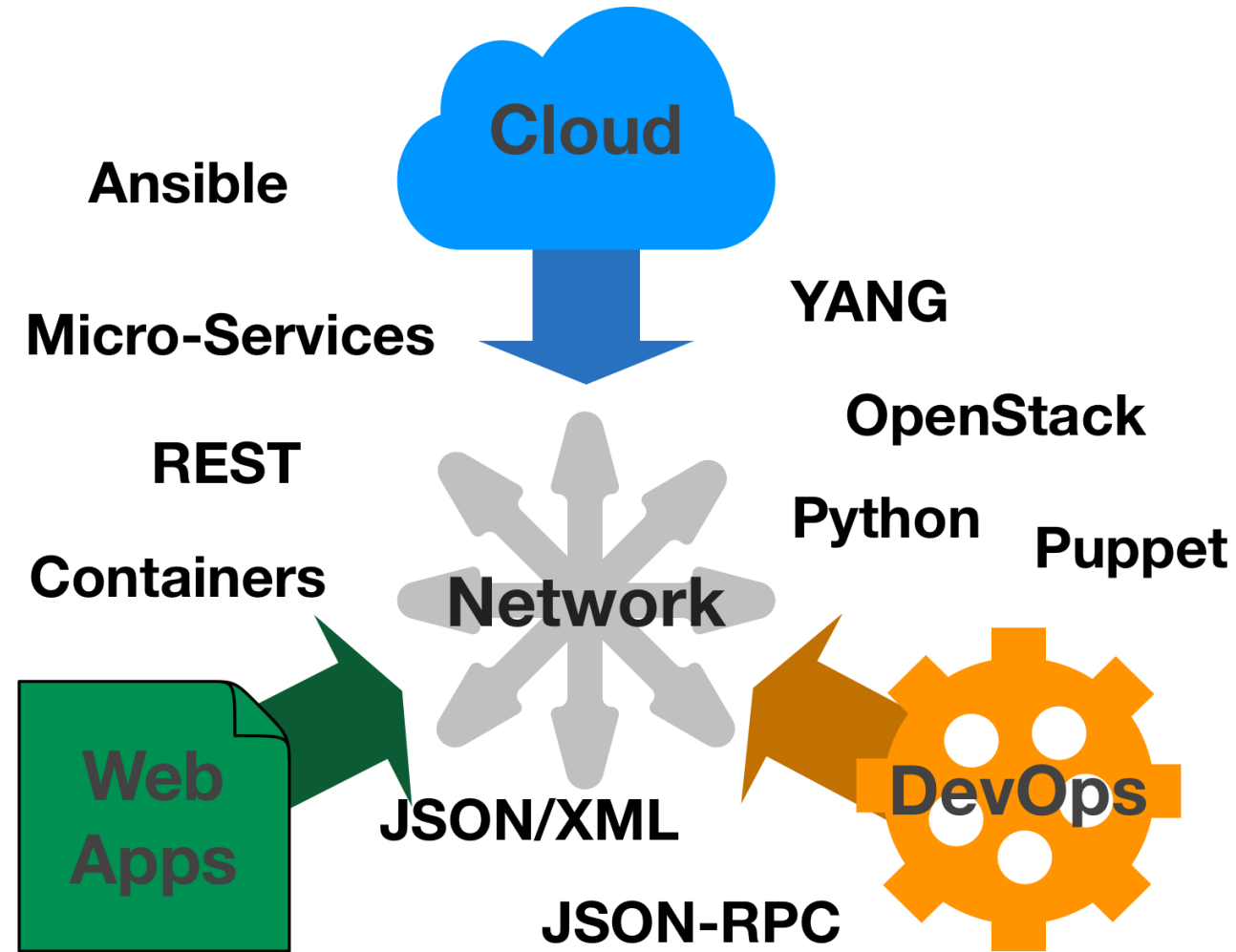
- The Road to Model Driven Programmability
- Introduction to YANG Data Models
- Introduction to NETCONF



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



# 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

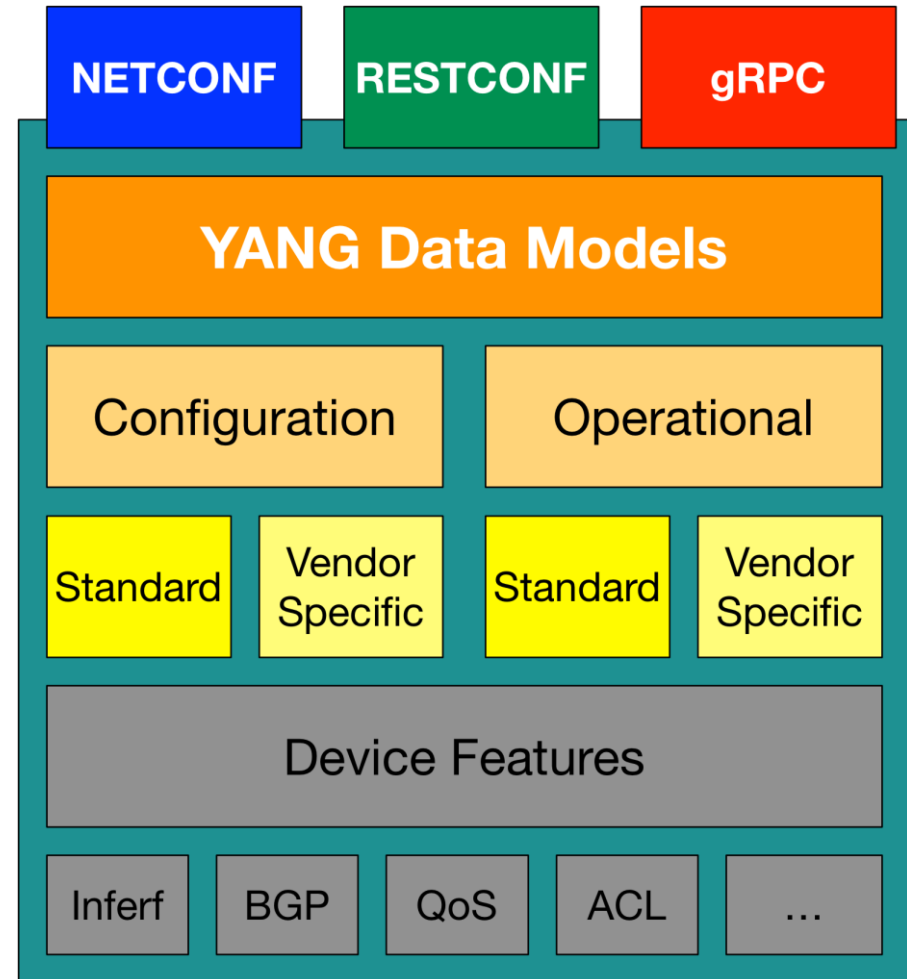
# RFC 3535: What is Needed?

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



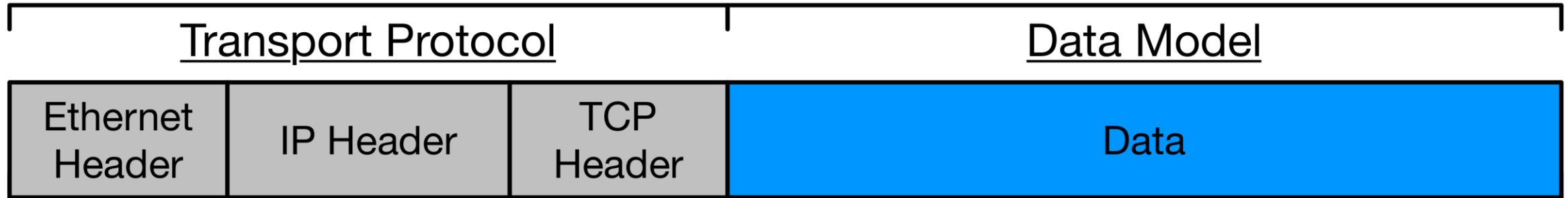
# Model Driven Programmability

- 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



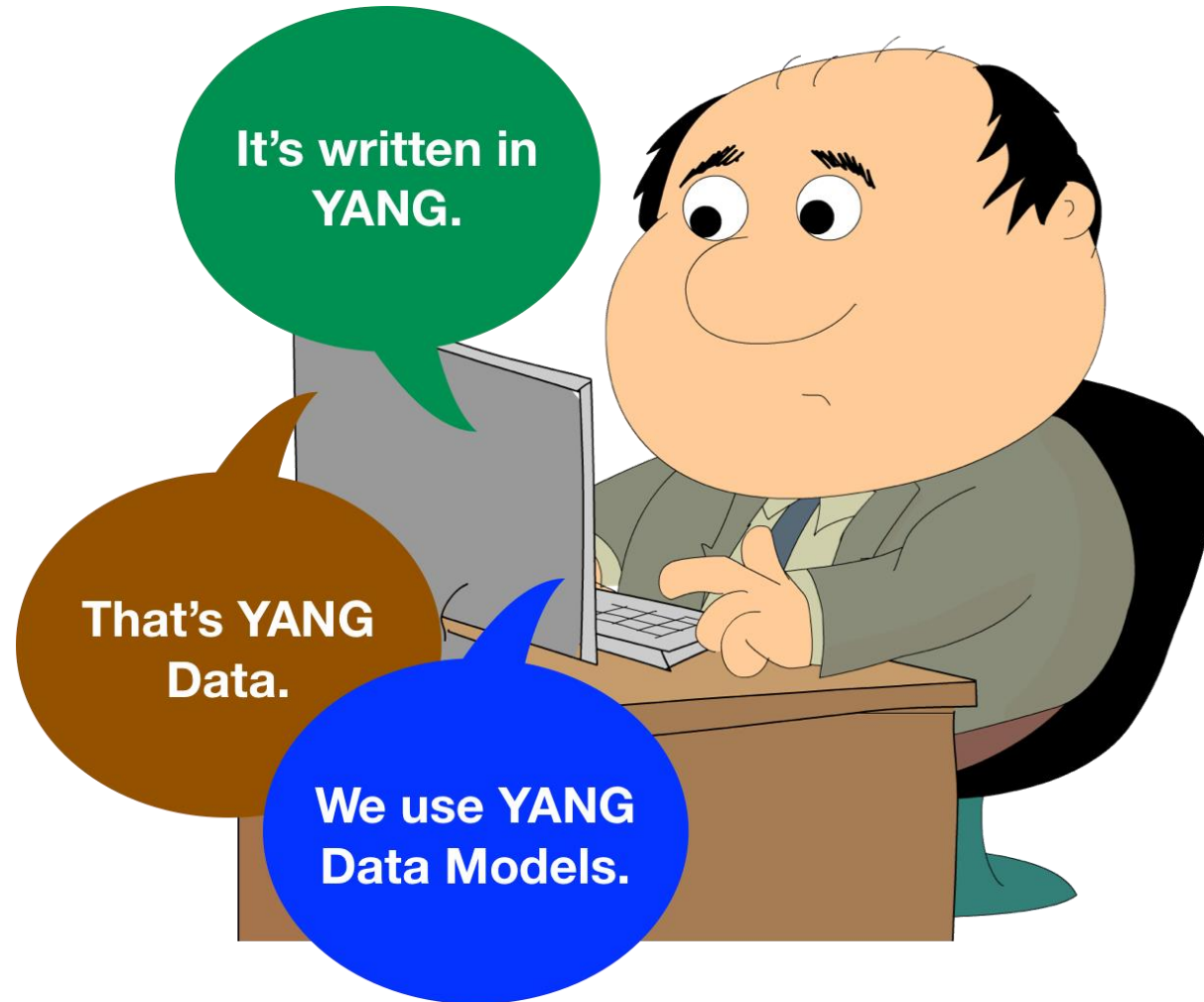
- NETCONF
- RESTCONF
- gRPC

- 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*

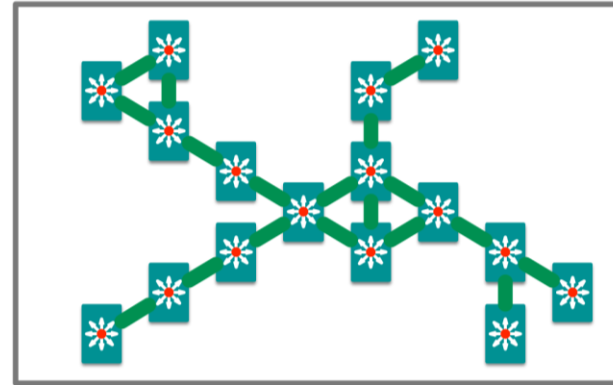
- **Gender** - male, female, other
- **Height** - Feet/Inches or Meters
- **Weight** - Pounds or Kilos
- **Hair Color** - Brown, Blond, Black, Red, other
- **Eye Color** - Brown, Blue, Green, Hazel, other

# 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-classifier.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>

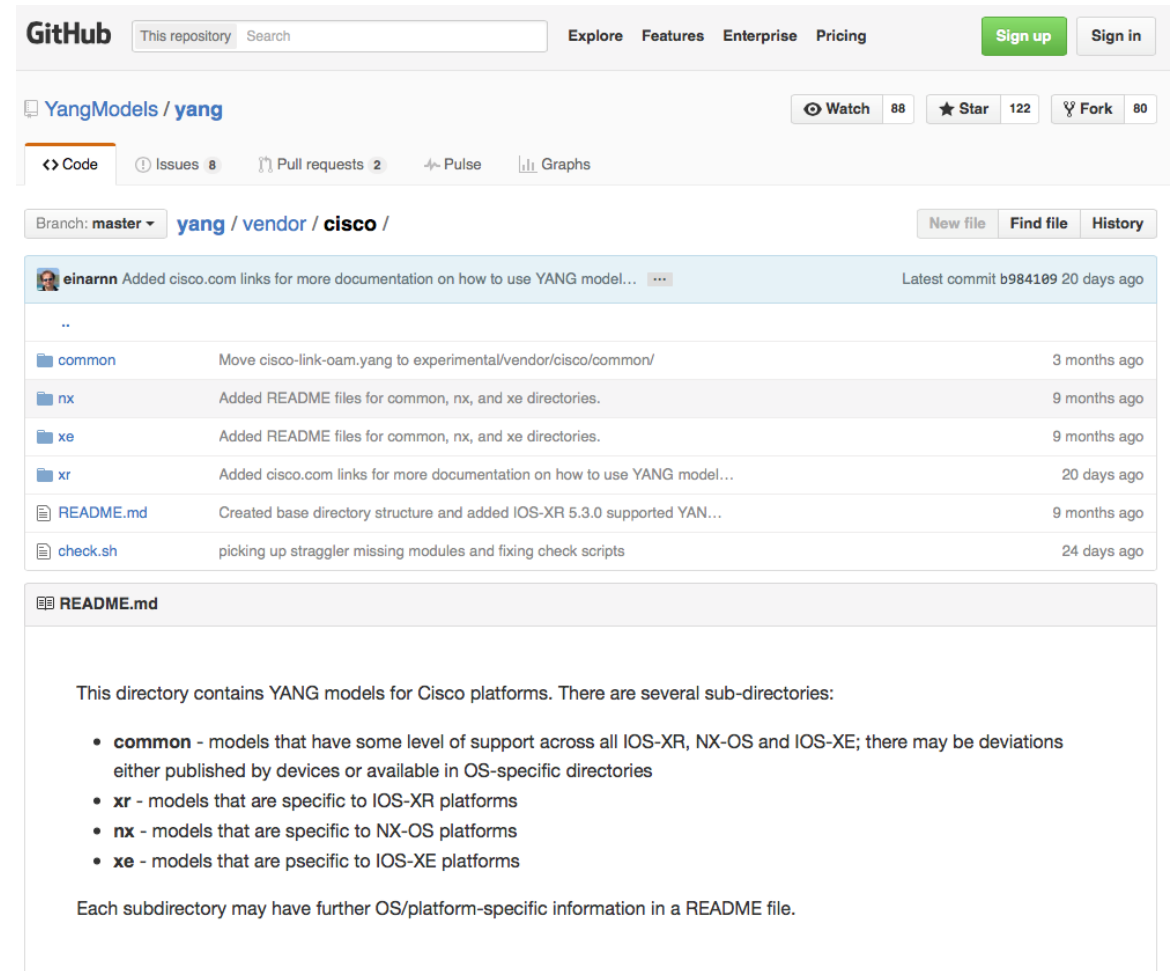
# Where to get the Models?

- 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>



The screenshot shows the GitHub repository page for `YangModels / yang`. The repository is on the `master` branch and is located at `yang / vendor / cisco /`. The latest commit by `einarmn` is titled "Added cisco.com links for more documentation on how to use YANG model..." and was made 20 days ago. The repository structure is as follows:

File/Directory	Description	Commit Date
..		
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

The `README.md` file contains the following text:

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](https://github.com/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
```

	Module Name
container	+--rw interfaces
	+--rw interface* [name] Key
	+--rw name string Leaf
	+--rw description? string
	+--rw type identityref
	+--rw enabled? Optional boolean
	+--rw link-up-down-trap-enable? enumeration {if-mib}?
container	+--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 Data Type
	+--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]

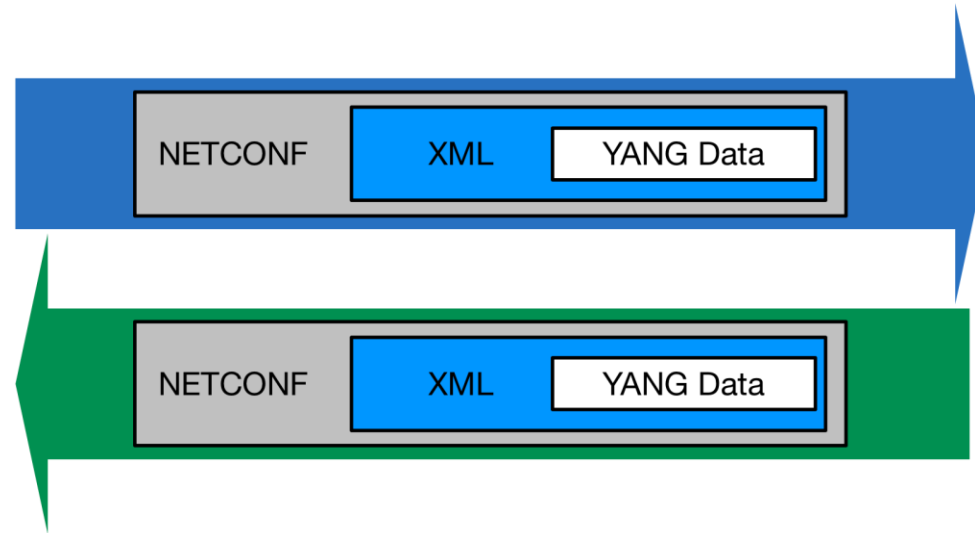
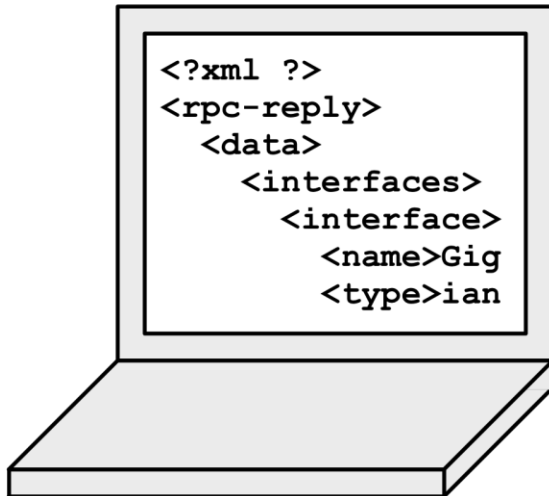
Example edited for simplicity and brevity

# Network Device Data in YANG

# Actual Device Data Modeled in YANG

## NETCONF Communications

### Manager



### Agent



# Use NETCONF to Retrieve ietf-interfaces data

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

```
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces"> Namespace = Capability = Model
  <interface>
    <name>GigabitEthernet1</name> Leaf
    <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>
</interfaces>
```

interfaces container



# YANG Summary

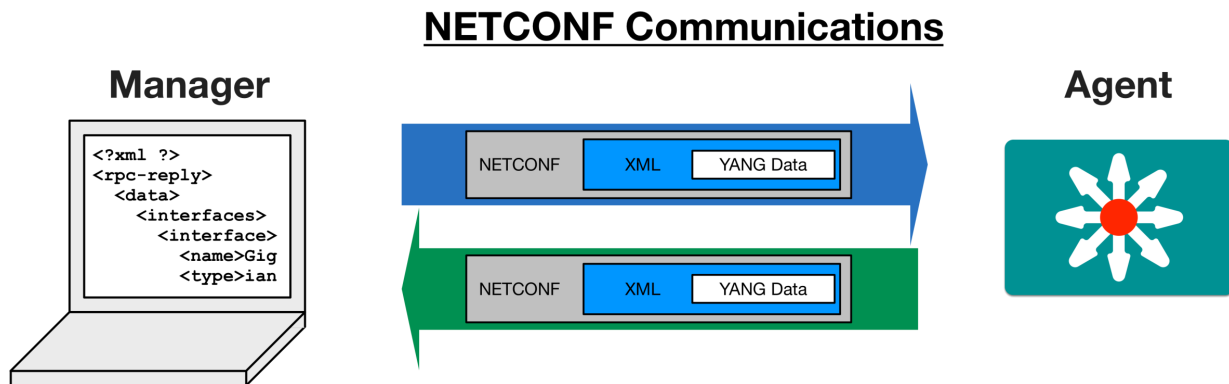
# YANG Summary

- 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

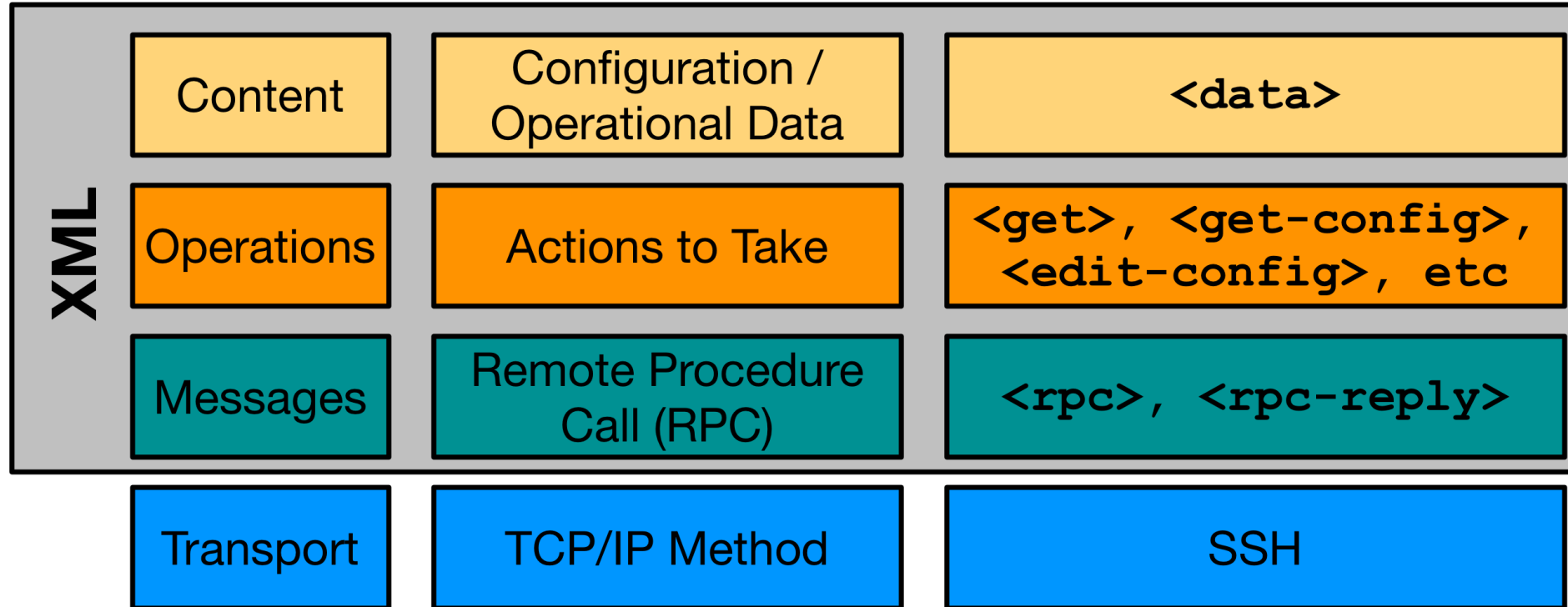
# Introducing the NETCONF Protocol



## Some key details:

- Initial standard in 2006 with [RFC4741](#)
- Latest standard is [RFC6241](#) in 2011
- Does **NOT** explicitly define content

# NETCONF Protocol Stack



# 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-transport</capability>  
  [output omitted and elided for clarity]  
</capabilities>  
<session-id>19150</session-id></hello>]]>]]>
```

Don't NETCONF Like this!

Server (Agent)  
sends hello

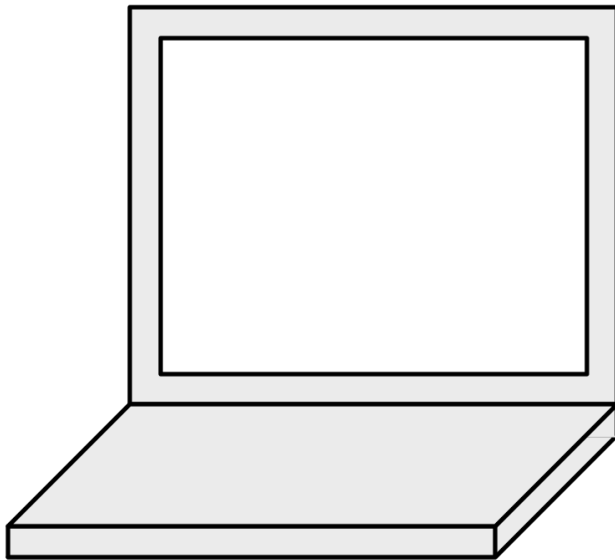
```
<?xml version="1.0" encoding="UTF-8"?>  
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">  
<capabilities>  
  <capability>urn:ietf:params:netconf:base:1.0</capability>  
</capabilities>  
</hello>]]>]]>
```

Client (Manager)  
sends hello

*Example edited for simplicity and brevity*

# Messages - Remote Procedure Call (RPC)

## Manager



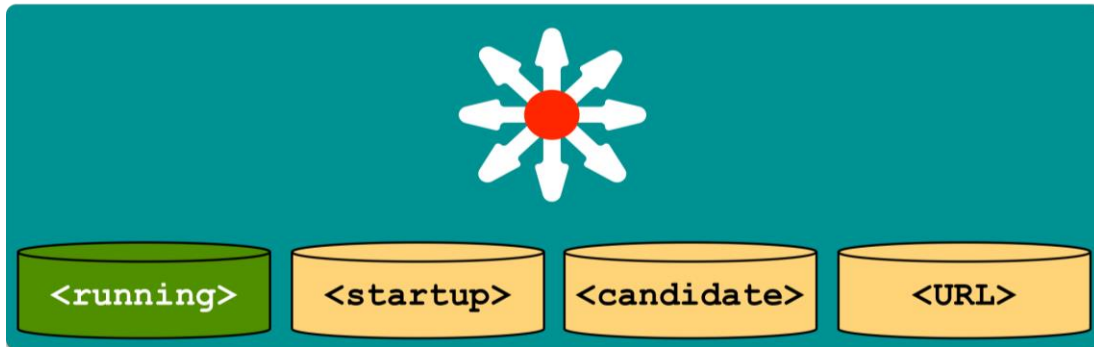
## Agent



# Operations – NETCONF Actions

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

# NETCONF Data Stores



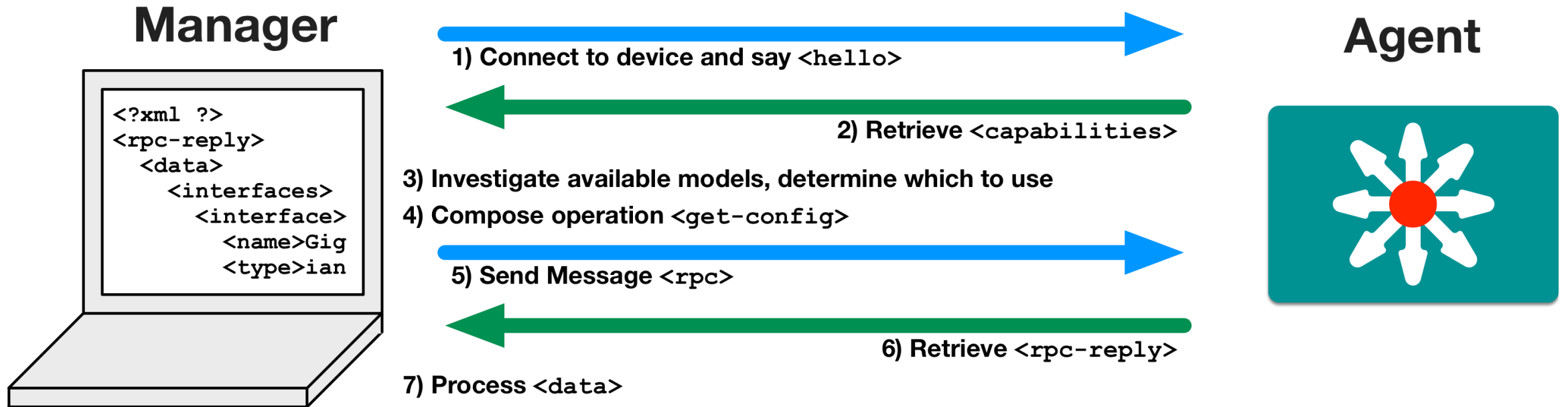
## Data Store Key Points

- Entire or partial configuration
- "running" is the only mandatory data store
- Not all data stores are writeable
- A "URL" data store is supported by IOS to enable <config-copy>
- Every NETCONF message must target a data store

```
result = m.get_config(' running ')
```



# NETCONF Communications



# 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](https://github.com/brkdev/BRKDEV-1368/blob/master/netconf/device_info.py)  
[BRKDEV-1368/netconf/example1.py](https://github.com/brkdev/BRKDEV-1368/blob/master/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](https://github.com/BRKDEV/1368/netconf/example2.py)  
[BRKDEV-1368/netconf/filter-ietf-interfaces.xml](https://github.com/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](https://github.com/BRKDEV/1368/netconf/example2.py)  
[BRKDEV-1368/netconf/filter-ietf-interfaces.xml](https://github.com/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](https://github.com/BRKDEV/1368/netconf/example2.py)  
[BRKDEV-1368/netconf/filter-ietf-interfaces.xml](https://github.com/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](https://www.cisco.com/BRKDEV-1368/netconf/example2.py)  
[BRKDEV-1368/netconf/filter-ietf-interfaces.xml](https://www.cisco.com/BRKDEV-1368/netconf/filter-ietf-interfaces.xml)

# Getting Interface Details with XPath

- example\_xpath.py: Retrieving info with ncclient and XPath
- Send <get> to retrieve and state data
- Process the data
- Report back current state of interface

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

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(filter=('xpath' ,
                                    "/interfaces-state/interface[name='GigabitEthernet1']"
                                    "/statistics/out-unicast-pkts"))

        [ intf_details = xmltodict.parse(netconf_reply.xml)["rpc-reply"]["data"]
          intf_info = intf_details["interfaces-state"]["interface"]

          [ print("")
            print("Interface Details:")
            print("  Name: {}".format(intf_info["name"]))
            print("  Packets Output: {}".format(intf_info["statistics"]["out-unicast-pkts"]))
```

```
DevNet$python example_xpath.py
```

```
Interface Details:
  Name: GigabitEthernet1
  Packets Output: 415200
```

[BRKDEV-1368/netconf/example\\_xpath.py](https://github.com/BRKDEV-1368/netconf/example_xpath.py)

# Configuring Interface Details

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

```
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](https://github.com/BRKDEV-1368/netconf/config-temp-ietf-interfaces.xml)

[BRKDEV-1368/netconf/example3.py](https://github.com/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](https://github.com/BRKDEV/1368/netconf/config-temp-ietf-interfaces.xml)

[BRKDEV-1368/netconf/example3.py](https://github.com/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

```
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](https://github.com/BRKDEV-1368/netconf/config-temp-ietf-interfaces.xml)

[BRKDEV-1368/netconf/example3.py](https://github.com/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>
```

[BRKDEV-1368/netconf/config-temp-ietf-interfaces.xml](https://github.com/devnetdeveloper/brkdev/blob/master/1368/netconf/config-temp-ietf-interfaces.xml)

[BRKDEV-1368/netconf/example3.py](https://github.com/devnetdeveloper/brkdev/blob/master/1368/netconf/example3.py)



# NETCONF Summary

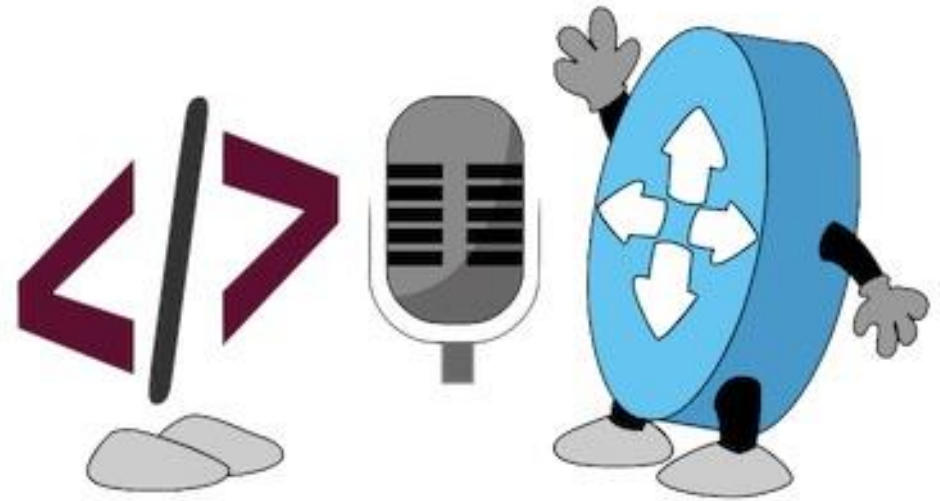
# NETCONF Summary

- 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

Summing up

# What did we talk about?

- The Road to Model Driven Programmability
- Introduction to YANG Data Models
- Introduction to NETCONF



# Webinar Resource List

- Docs and Links
  - <https://developer.cisco.com/netconf>
- Learning Labs
  - Model Driven Programmability <http://cs.co/lab-mdp>
  - NETCONF/YANG on Nexus <http://cs.co/lab-mdp-nexus>
- DevNet Sandboxes
  - IOS Always On <http://cs.co/sbx-iosxe>
  - NX-OS Always On <http://cs.co/sbx-nxos>
  - IOS XR Reserved <http://cs.co/sbx-iosxr>
- Code Samples
  - <https://github.com/CiscoDevNet/BRKDEV-1368>

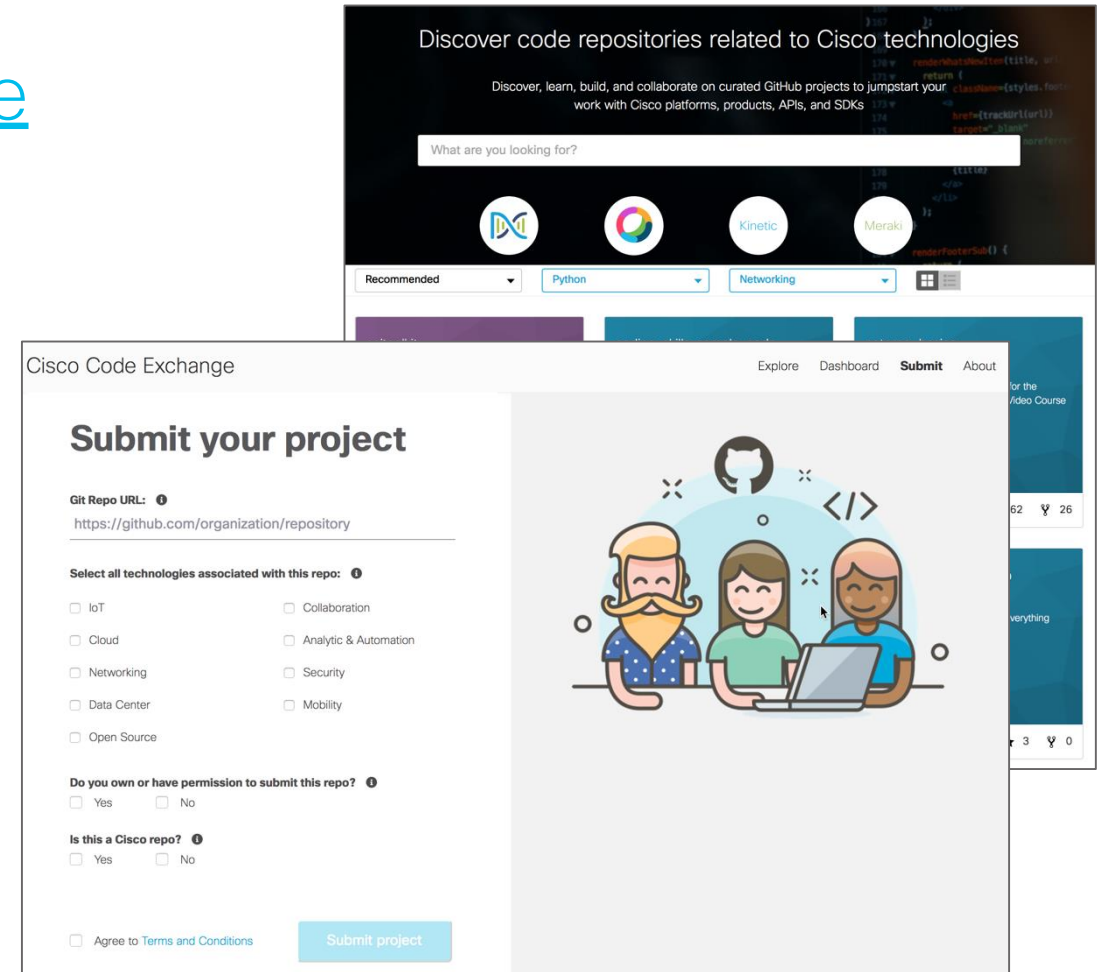


# NetDevOps Live! Code Exchange Challenge

[developer.cisco.com/codeexchange](https://developer.cisco.com/codeexchange)

**Use NETCONF to configure basic routing using your favorite protocol.**

*Hint: Configure the device with CLI the first time, and use `m.get_config("running")` to retrieve the NETCONF configuration to build a template.*



The image shows two overlapping screenshots of the Cisco Code Exchange website. The top screenshot is a search page with the heading "Discover code repositories related to Cisco technologies" and a search bar. Below the search bar are icons for various technologies: Cisco, Python, Kinetic, and Meraki. The bottom screenshot is the "Submit your project" form, which includes a "Git Repo URL" field, a "Select all technologies associated with this repo" section with checkboxes for IoT, Cloud, Networking, Data Center, Open Source, Collaboration, Analytic & Automation, Security, and Mobility, and a "Submit project" button.

# Looking for more about NetDevOps?

- NetDevOps on DevNet  
[developer.cisco.com/netdevops](https://developer.cisco.com/netdevops)
- NetDevOps Live!  
[developer.cisco.com/netdevops/live](https://developer.cisco.com/netdevops/live)
- NetDevOps Blogs  
[blogs.cisco.com/tag/netdevops](https://blogs.cisco.com/tag/netdevops)
- Network Programmability Basics Video Course  
[developer.cisco.com/video/net-prog-basics/](https://developer.cisco.com/video/net-prog-basics/)



Got more questions? Stay in touch!



**Bryan Byrne**



brybyrne@cisco.com



@brybyrne25607



<http://github.com/brybyrne>



**DEVNET**

LEARN CODE INSPIRE CONNECT

**[developer.cisco.com](http://developer.cisco.com)**



@CiscoDevNet



[facebook.com/ciscocodevnet/](https://facebook.com/ciscocodevnet/)

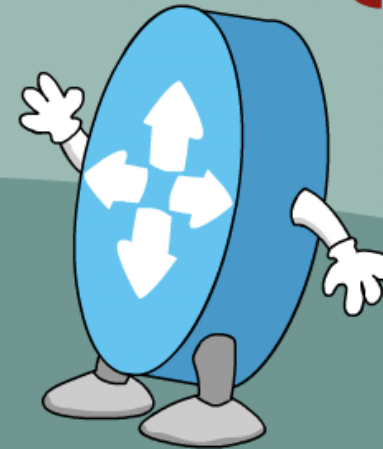


<http://github.com/CiscoDevNet>





# NETDEVOPS {LIVE!}



DEVNET

<https://developer.cisco.com/netdevops/live>

@netdevopslive 