pue
IMPULSANDO EL CONOCIMIENTO
TIC CUALIFICADO

EDUCACIÓN
CALIDAD Y CERTIFICACIÓN
EN SUS PROGRAMAS EDUCATIVOS

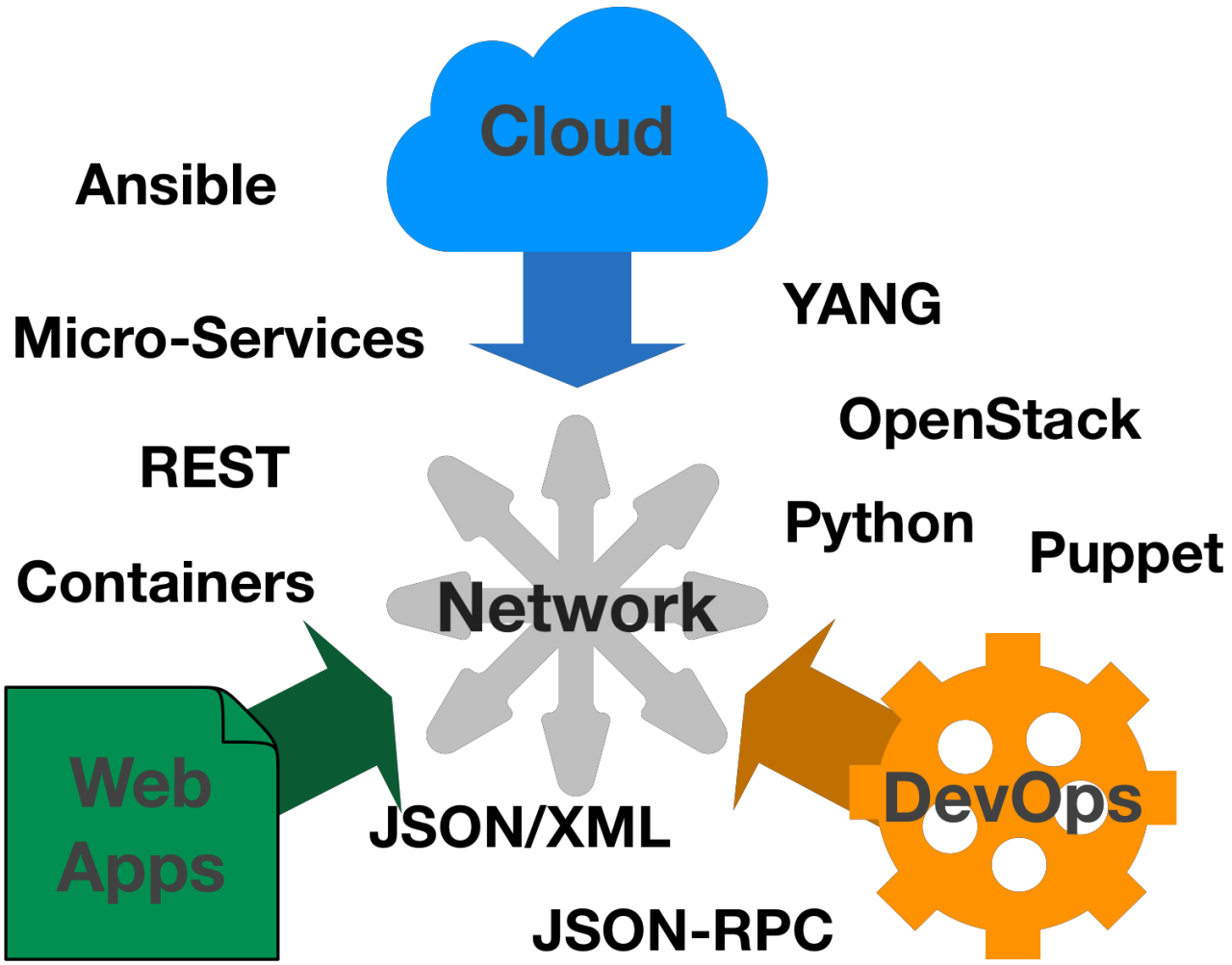
Programación de Redes – Becas Digitaliza - 2019
PUE – ITC – Formación de Instructores
Sesión 9 – NETCONF + YANG

Iván Lago - Técnico Cisco Networking Academy ASC/ITC
PUE - ITC/ASC/CA

Index

Python for Network Engineers

The Network is no longer isolated



What about SNMP?

SNMP works "reasonably well for device monitoring"

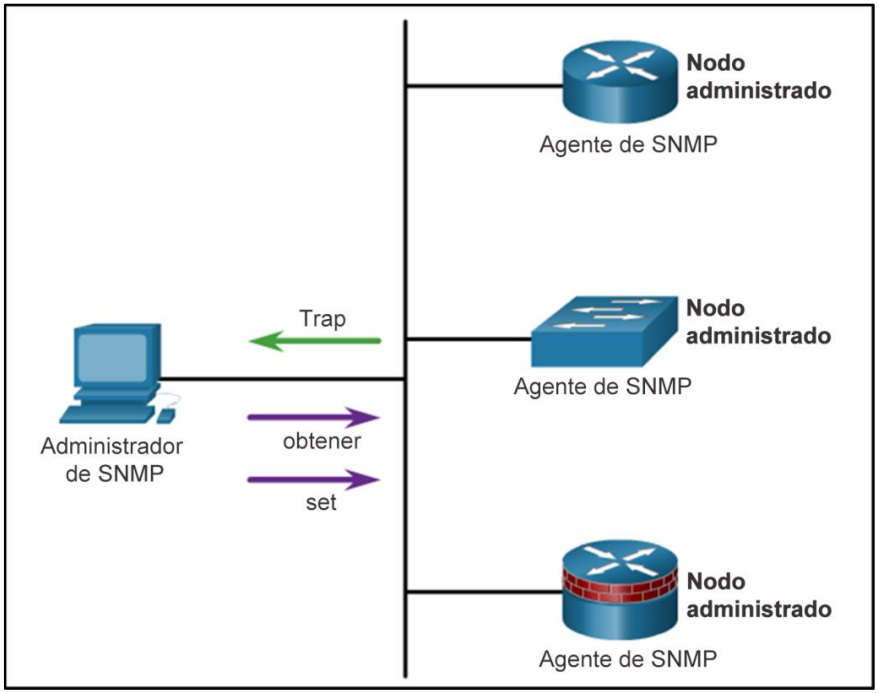
- Typical config: SNMPv2 read-only community strings
- Typical usage: interface statistics queries and traps
- Empirical Observation: SNMP is not used for configuration
  - Lack of Writeable MIBs
  - Security Concerns
  - Difficult to Replay/Rollback
  - Special Applications

Operación de SNMP

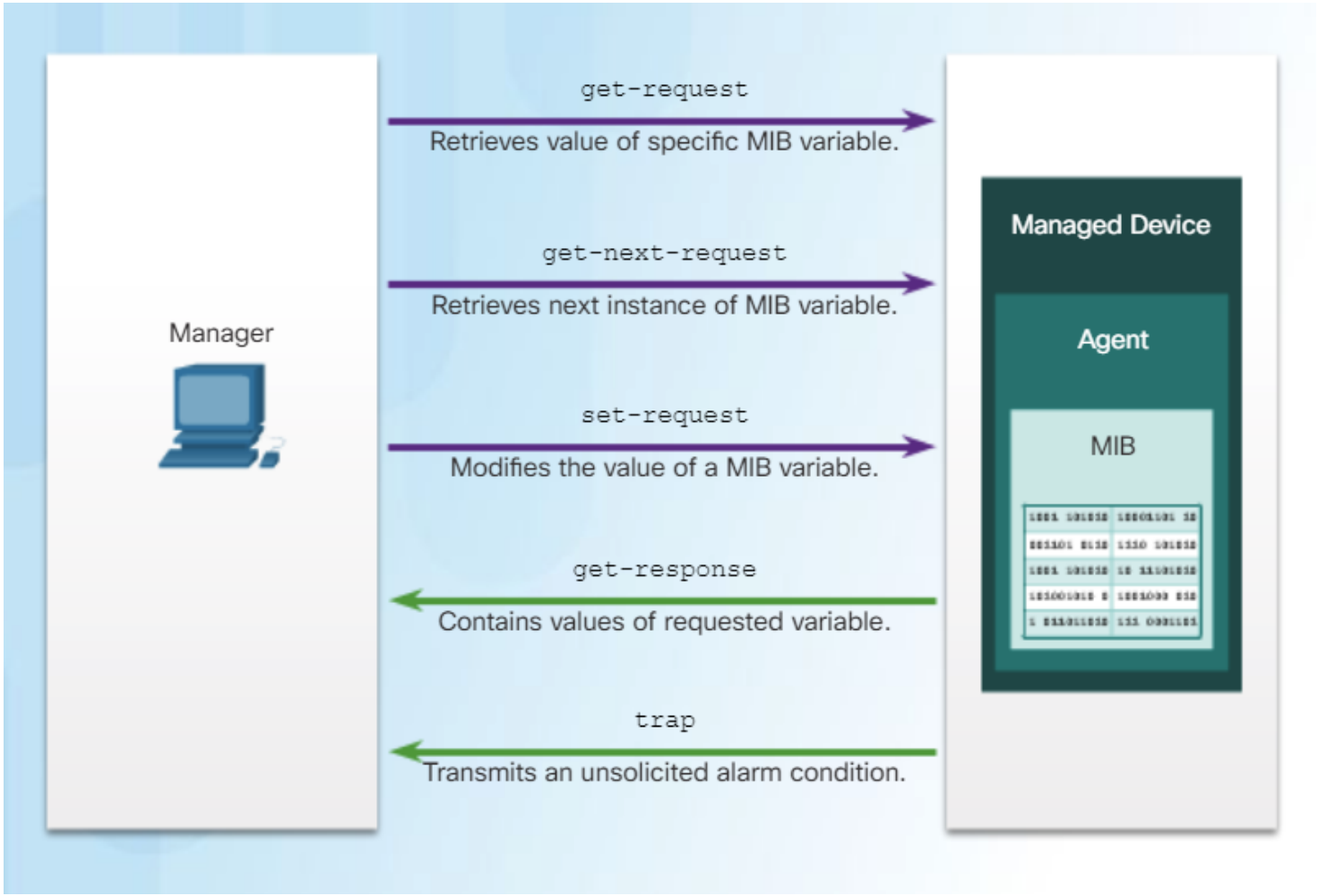
- El SNMP permite que los administradores administren y monitoreen dispositivos en una red IP.

- Elementos de SNMP
  - Administrador de SNMP
  - Agente de SNMP (nodo)
  - MIB (Management information base)

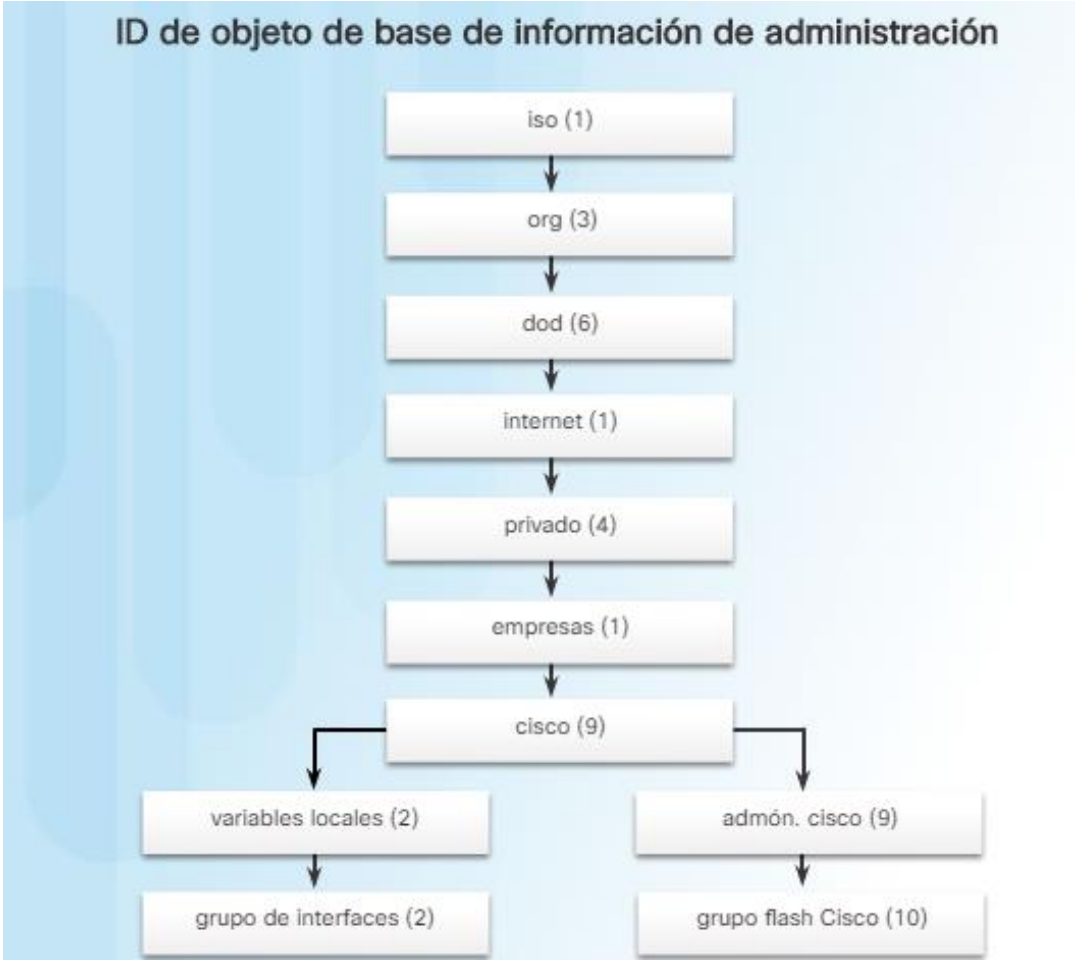
- Funcionamiento de SNMP
  - trap, get y set



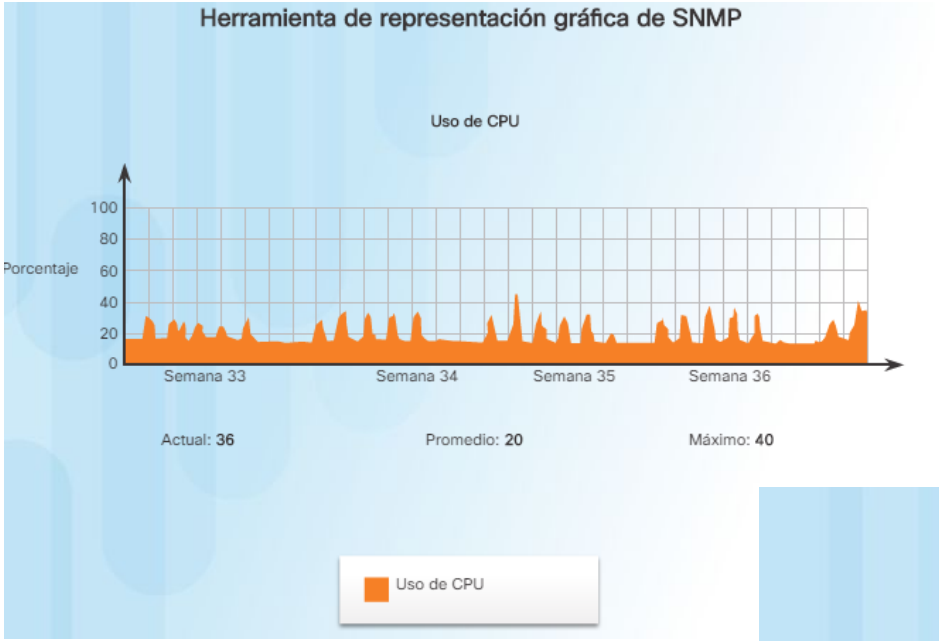
Interacciones SNMP

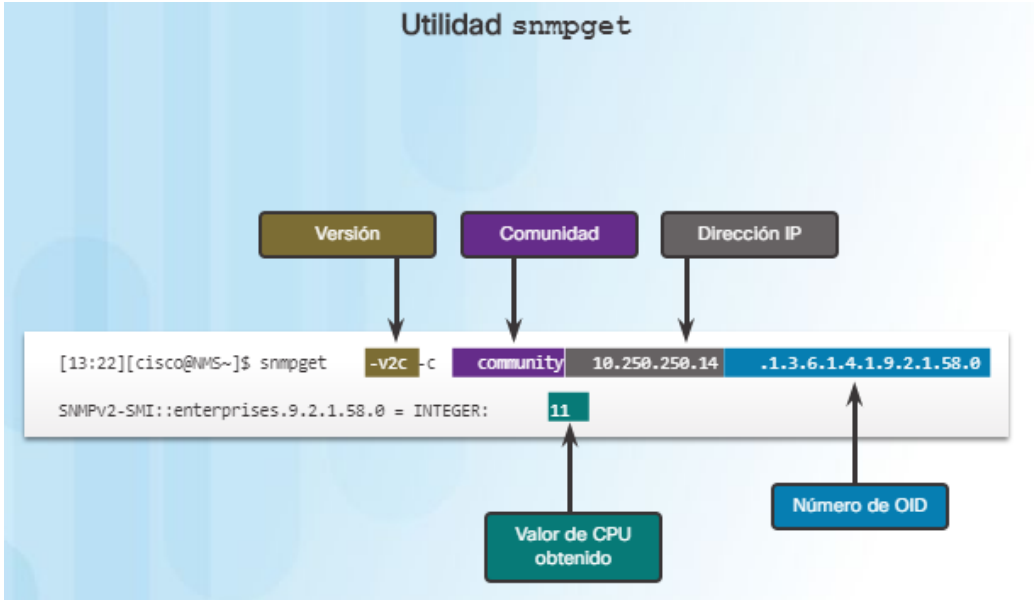


MIB: Management Information Base



Monitorización + MIB





Versiones de SNMP

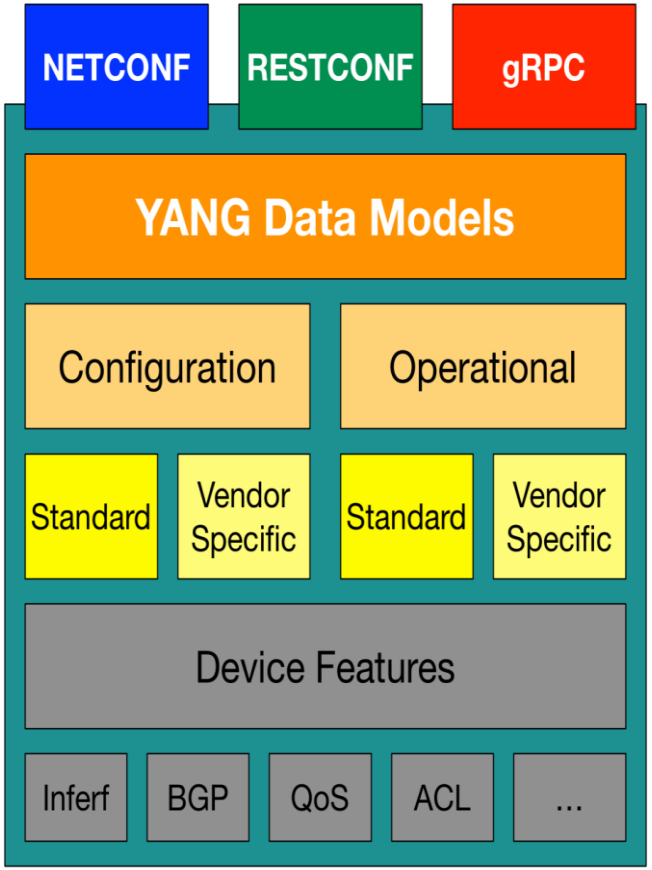
Modelo	Nivel	Autenticación	Cifrado	Resultado
SNMPv1	noAuthNoPriv	Cadena de comunidad	No	Usa una coincidencia de cadena de comunidad para la autenticación.
SNMPv2c	noAuthNoPriv	Cadena de comunidad	No	Usa una coincidencia de cadena de comunidad para la autenticación.
SNMPv3	noAuthNoPriv	Nombre de usuario	No	Usa una coincidencia de nombre de usuario para la autenticación (una mejora con respecto a SNMPv2c).
SNMPv3	authNoPriv	Algoritmo de síntesis del mensaje 5 (MD5) o algoritmo hash seguro (SHA)	No	Proporciona autenticación basada en los algoritmos HMAC-MD5 o HMAC-SHA.
SNMPv3	authPriv (requiere la imagen del software criptográfico)	MD5 o SHA	Estándar de cifrado de datos (DES) o estándar de cifrado avanzado (AES)	Proporciona autenticación basada en los algoritmos HMAC-MD5 o HMAC-SHA.

What do we need?

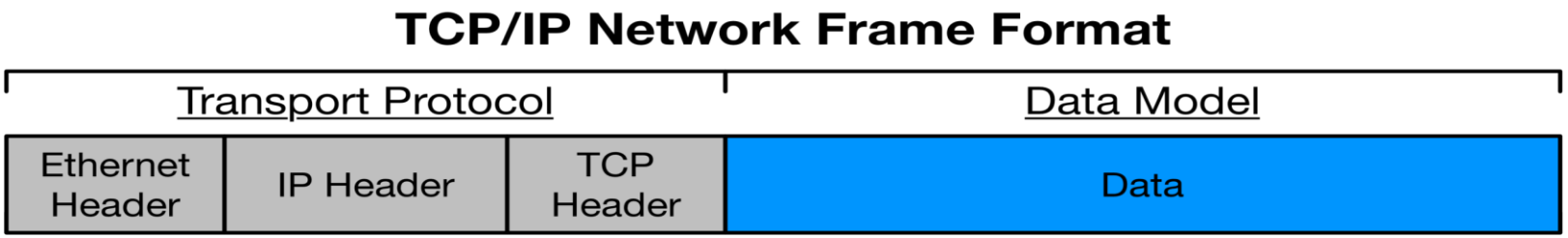
- A programmatic interface for device configuration
- Separation of Configuration and State Data
- Ability to configure "services" NOT "devices"
- Integrated error checking and recovery

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)

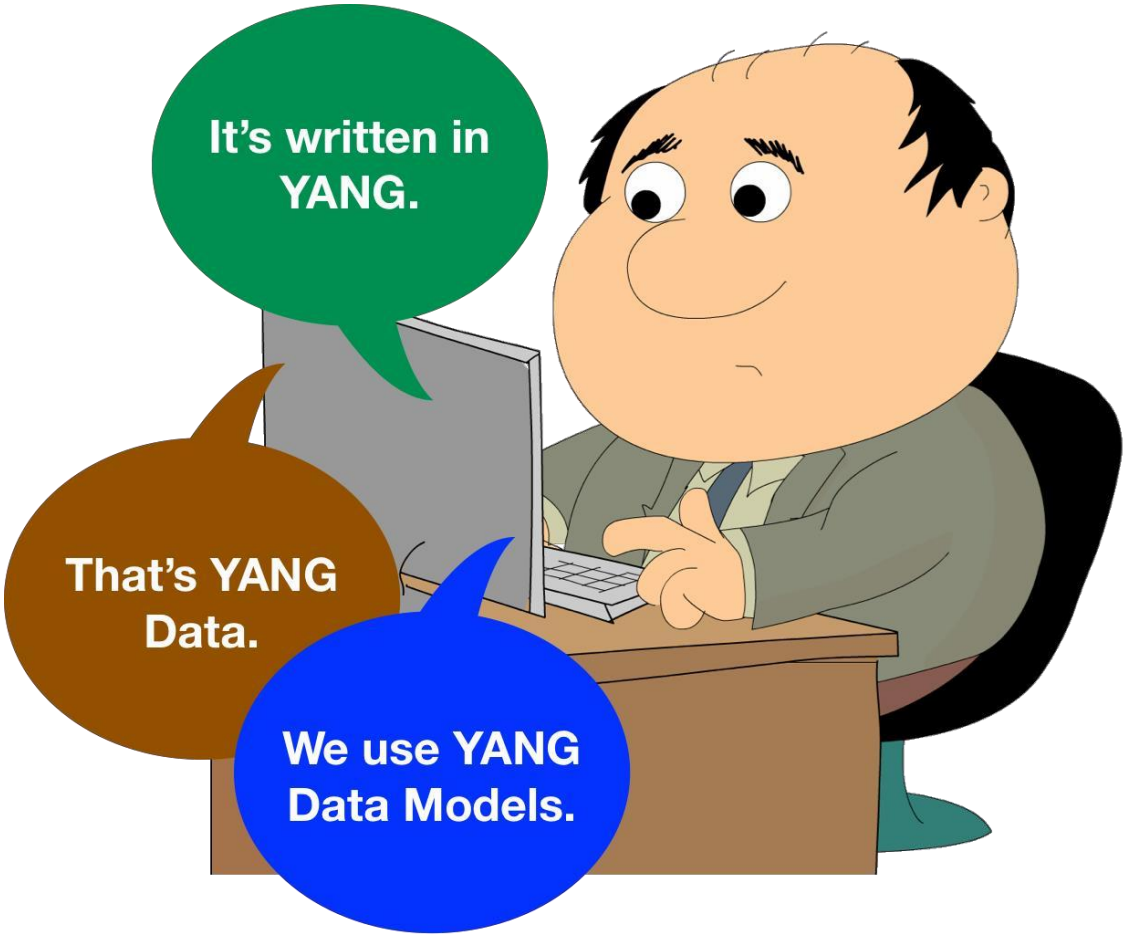


- NETCONF
- RESTCONF
- gRPC

- YANG

What is YANG?

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";
    }
}
```

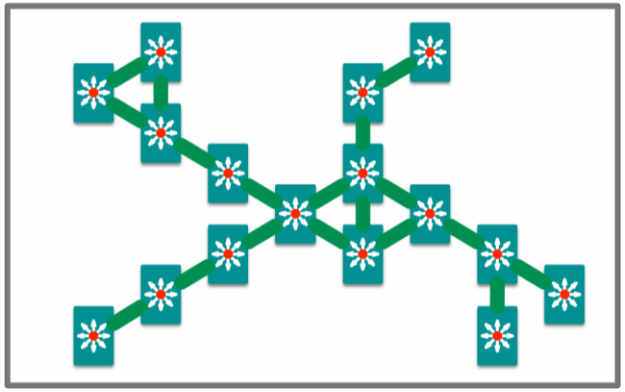
What is a Data Model?

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.

Example:

- 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?

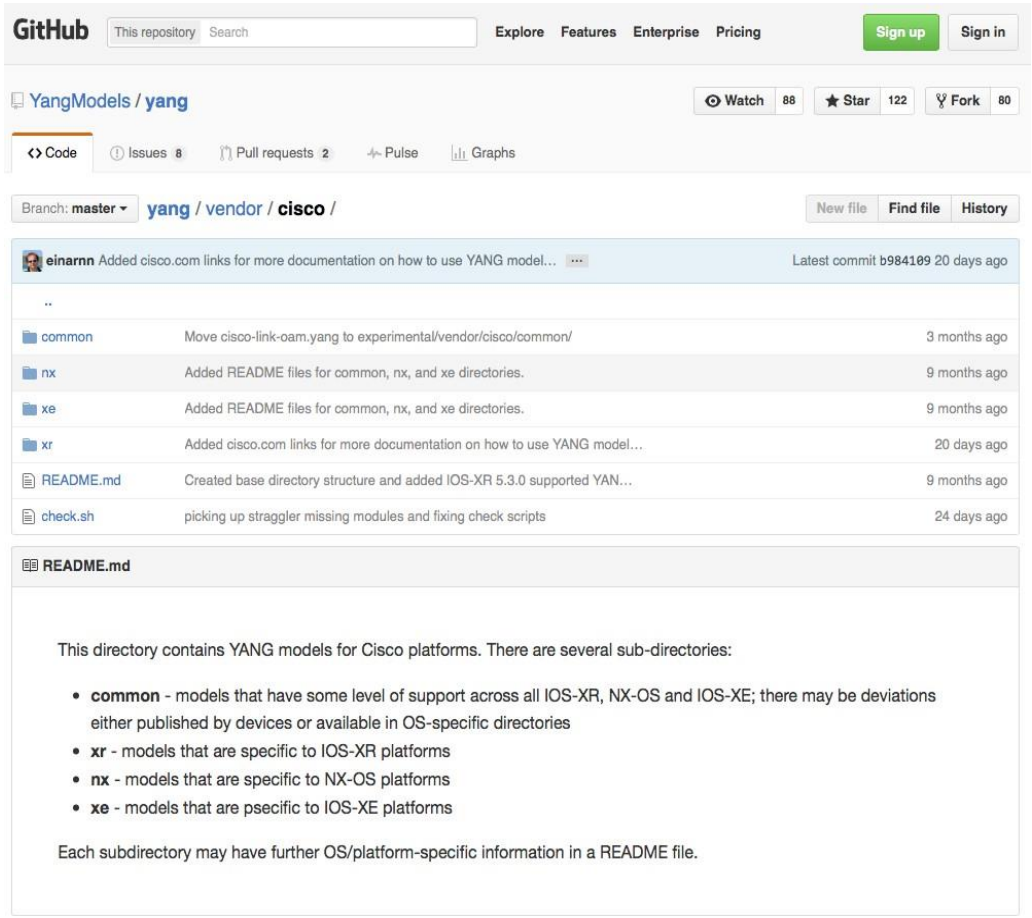
- **Standard definition**
  (IETF, ITU, OpenConfig, etc.)
- Compliant with standard
  `ietf-diffserv-policy.yang`
  `ietf-diffserv-classifer.yang ietf-diffserv-target.yang`

- **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

Where to get the Models?

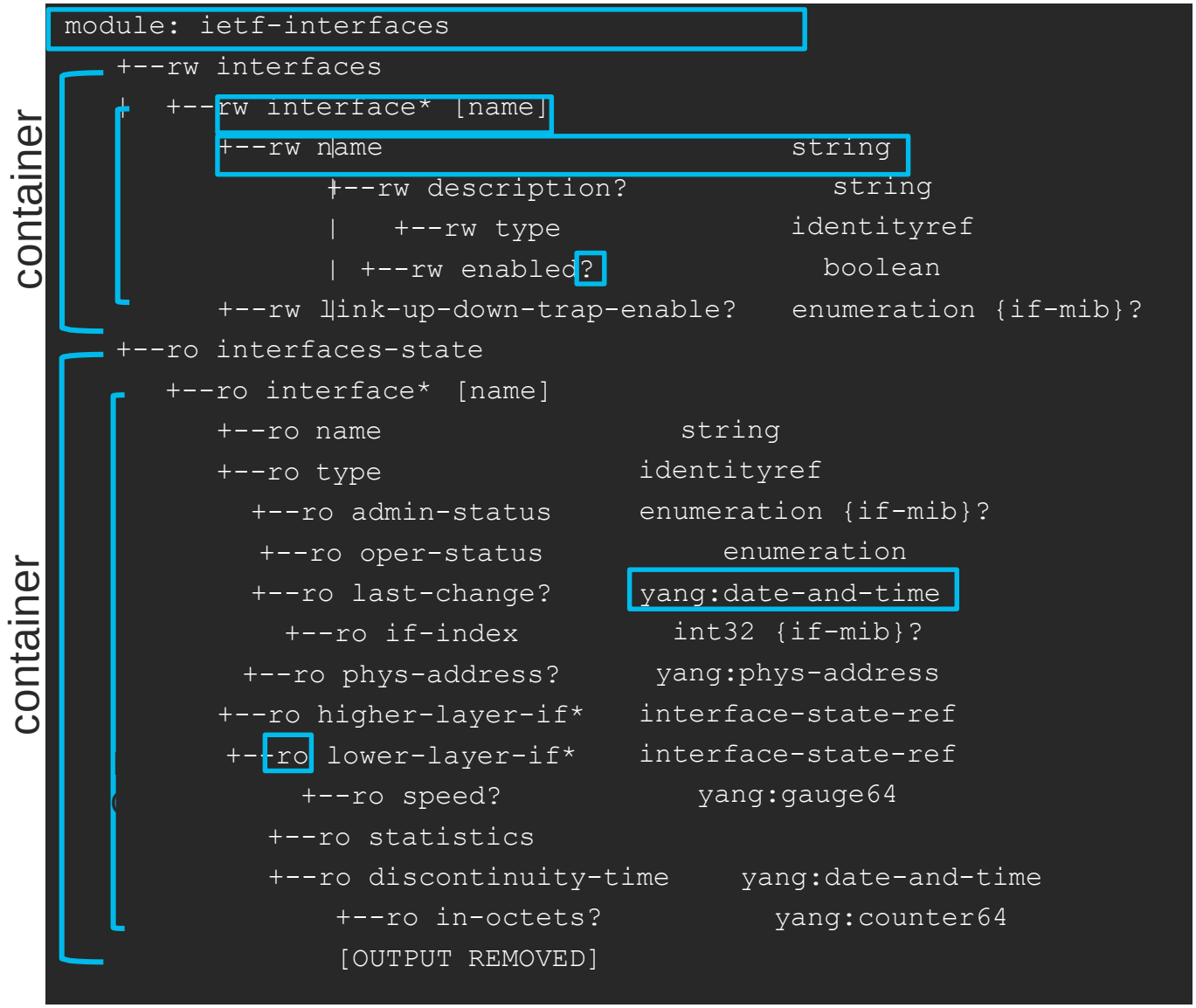
- For YANG modules from standard organizations such as the IETF, open source such as Open Daylight or vendor specific modules”
  - https://github.com/YangModels/yang
- For OpenConfig models
  - https://github.com/openconfig/public



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
- To work with YANG we should install the pyang library
  - pip install pyang
- We have to use YANG along with transport protocols, such as NETCONF (only supports XML) or RESTCONF (XML or JSON)

Working with YANG Data Models (I)

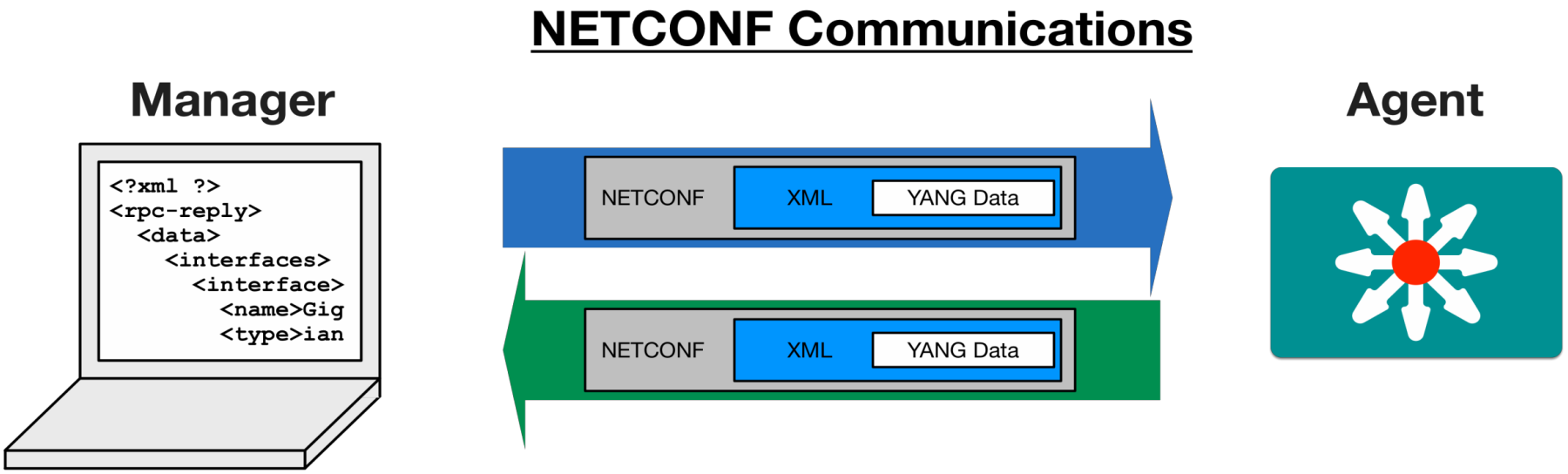


Working with YANG Data Models (II)

- There are two containers: "rw interfaces" and "ro interfaces-state“ --> Each contanier has a list of objects
- The interface name is the "key" to distinguish one interface from another.
- Each individual attribute inside an object is a leaf. (in the first faces, leafs are name, description, type, enabled and link-up-down-trap-enable)
- Objects may be read only (ro) or read write (rw)
- "?" -> indicates that may be optional attributes
- date and time -> every object has a data type

Introduction to NETCONF

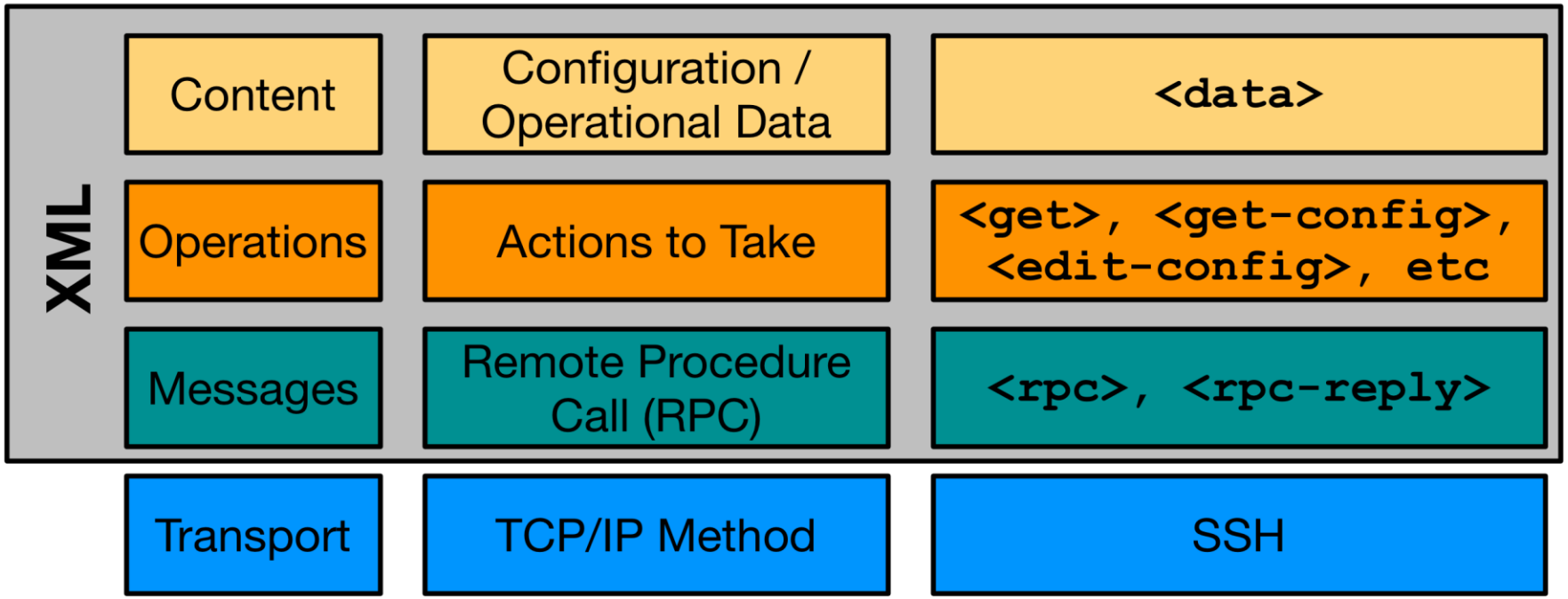
Actual Device Data Modeled in YANG



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>]]>]]>
```

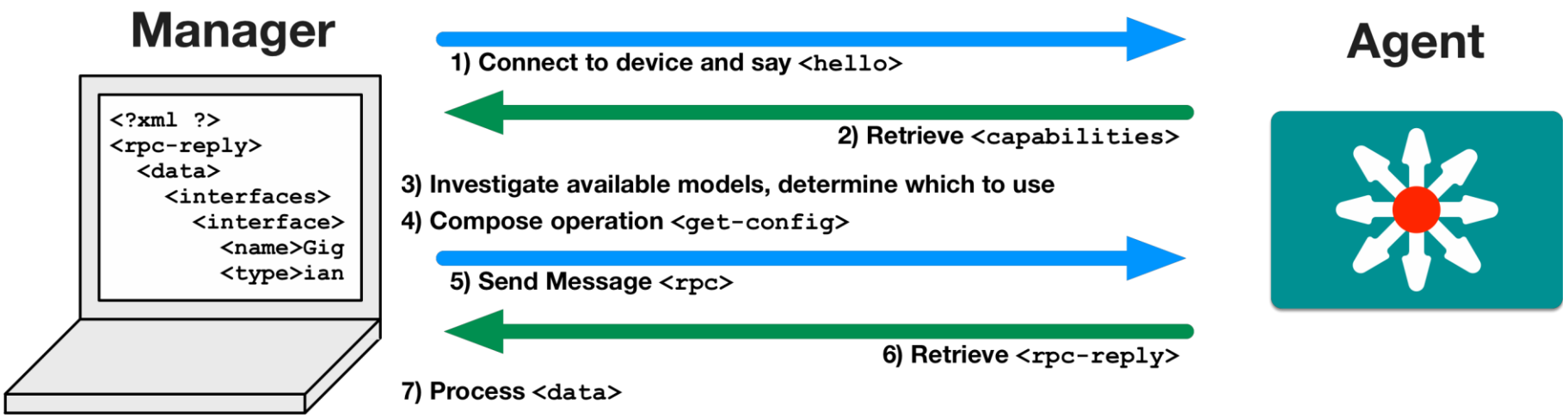
Client (Manager) sends hello

DON’T DO IT IN THIS WAY!

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



rpc -> Remote Procedure Call

NETCONF: getting capabilites

netconf1.py

Understanding the Capabilities List (I)

```
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
```

Two General Types

- Base NETCONF capabilities
- Data Models Supported

Understanding the Capabilities List (II)

```
    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
```

Data Model Details

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

NETCONF in Code with Python

NETCONF: capabilities

- ncclient --> pip install ncclient
- Remember that NETCONF deals with XML
- https://devnetsandbox.cisco.com/RM/Diagram/Index/27d9747a-db48-4565-8d44-df318fce37ad?diagramType=Topology
- Example: netconf1.py (see capabilities –two general types)
  - Base NETCONF capabilities
  - Data Models Supported
    - URI -> *urn:ietf:params:xml:ns:yang:ietf-interfaces*
    - Module Name and Revision Date -> *module=ietf-interfaces revision=2014-05-08*
    - Protocol Features -> *features=pre-provisioning, if-mib, arbitrary-names*
    - Deviations (another model that modifies this one) -> *deviations=ietf-ip-devs*

NETCONF: getting info and configuring

netconf2.py

Getting Interface Details with XML Filter (I)

- 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"]))
```

Getting Interface Details with XML Filter (II)

- 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>
```

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
```

NETCONF: getting info and configuring

netconf3.py

Configuring Interface Details (I)

- 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)
```

Configuring Interface Details (II)

- 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)
```

Configuring Interface Details (III)

```
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>
```

DEVNET
developer.cisco.com

Gracias por vuestra atención

Iván Lago - Técnico Cisco Networking Academy ASC/ITC
PUE - ITC/ASC/CA
Área de Proyectos de Educación