Configure gNMI and Implement pYANG in IOS XR

Contents

Introduction

This document describes a brief on gNMI in Cisco IOS® XR and how to use PYANG and check model trees.

Prerequisites

Requirements

Cisco recommends that you have knowledge of these topics:

- Cisco IOS XR platform.
- python.
- Network Management Protocols.

Components Used

This document is not restricted to specific hardware versions it apply to 64-bits version (eXR).

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, ensure that you understand the potential impact of any command.

Background Information

gNMI definition

Overall there are different Network Configuration Protocols, from NETCONF, RESTCONF, gNMI (Google Remote Procedure Calls (gRPC), gRPC Network Management Interface), inter alia. These models are used to configure to manage the network devices and always aim to automate processes that can be mechanic.

These protocols utilize different data models to allow the users to understand what the network device process, in other words, it is a structured information, a schema, that normalize information and how it is consumed by the device, in this case, the router.

gNMI oversees the data handing and provide RPC (Remote Procedure Calls) to control the different devices in the network.

gNMI has four Functions:

- Capabilities: gNMI ask the router the models that are installed in the router, this is explained further in this document.
- Get: Every leaf component in the data tree can be requested to the router, this operation request the information requested.
- Set: Leafs are consider as variables, what provides them with the change capabilities, Set operation assist on this allowing the user to update a value in the data model.
- Subscribe: Utilized in Telemetry, this function assist in pulling data from a particular module in the model.

Note: Cisco has shared a lot of information on this topic. For further information from gRPC, click the next link: xrdocs blog - OpenConfig gNMI

---

gNMI features

Network Management Protocol	gNMI
Transport utilized	HTTP/2
Supported by	Vendor Neutral
Encoding	Proto Buff

Proto Buff is the language-neutral, platform-neutral method of de-serializing and serializing data between two devices, in which, each Request have a Reply.

Note: For more details in gRCP and Proto Buff click the next link: grpc Guide.

gNMI Basic Configuration in Cisco IOS XR

The next is the basic configuration for the router:

```
RP/0/RSP0/CPU0:XR(config)#grpc
RP/0/RSP0/CPU0:XR(config-grpc)#address-family ipv4
RP/0/RSP0/CPU0:XR(config-grpc)#max-request-total 256
RP/0/RSP0/CPU0:XR(config-grpc)#max-request-per-user 32

grpc
 address-family ipv4
 max-request-total 256
 max-request-per-user 32
```

Note: A port can be configured based on the setup, the default, without using TLS is 57400, for more information click: github - grpc getting started

pYANG as validator

pYANG is a YANG validator written in python. This library for python assist on checking the YANG models and also, knowing them.

For this to run as in the documentation (pYANG documentation) it is suggested to create a virtual environment in the computer.

For virtual environment to run venv documentation

It is required to run:

```
python -m venv <name of the directory>
```

For example (in MacOS terminal):

```
% mkdir test
% cd test
% python3 -m venv virtual_env
% ls
virtual_env
```

To install pYANG in this virtual environment cd to the directory and paste the next:

```
% cd virual_env
% git clone https://github.com/mbj4668/pyang.git
% cd pyang
% pip install -e .
```

For this demonstration, python3 pip was used, once the **pip install -e** is issued, activate the venv: **source <virtual environment directory>/bin/activate** (for MacOS).

```
% source virtual_env/bin/activate
```

```
% python3 -m pip install pyang
Collecting pyang
  Downloading pyang-2.6.0-py2.py3-none-any.whl (594 kB)
     |████████████████████████████████| 594 kB 819 kB/s
Collecting lxml
  Downloading lxml-5.1.0-cp39-cp39-macosx_11_0_arm64.whl (4.5 MB)
     |████████████████████████████████| 4.5 MB 14.2 MB/s
Installing collected packages: lxml, pyang
Successfully installed lxml-5.1.0 pyang-2.6.0
```

```
% pyang -h
Usage: pyang [options] [<filename>...]

Validates the YANG module in <filename> (or stdin), and all its dependencies.

Options:
  -h, --help            Show this help message and exit
  -v, --version         Show version number and exit
<snip>
```

With pYANG installed and working, continue with models download.

In the next link there are all the models that Cisco IOS XR run: Cisco IOS XR models.

It is suggested to **git clone** this models in the venv directory with the next code link: https://github.com/YangModels/yang.git

---

Note: This is not done with the virtual environment activated.

```
% git clone https://github.com/YangModels/yang.git
```

```
Cloning into 'yang'...
remote: Enumerating objects: 54289, done.
remote: Counting objects: 100% (1910/1910), done.
remote: Compressing objects: 100% (323/323), done.
remote: Total 54289 (delta 1643), reused 1684 (delta 1586), pack-reused 52379
Receiving objects: 100% (54289/54289), 116.64 MiB | 8.98 MiB/s, done.
Resolving deltas: 100% (42908/42908), done.
Updating files: 100% (112197/112197), done.
```

Activate the virtual environment again and test the next query: **pyang -f tree yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang**.

```
(virtual_env) % pyang -f tree yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang
yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang:5: error: module "Cisco-IOS-XR-types" not found in
yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang:8: error: module "cisco-semver" not found in search
module: Cisco-IOS-XR-ifmgr-cfg
  +--rw global-interface-configuration
  |  +--rw link-status?   Link-status-enum
  +--rw interface-configurations
     +--rw interface-configuration* [active interface-name]
        +--rw dampening
        |  +--rw args?                 enumeration
        |  +--rw half-life?            uint32
        |  +--rw reuse-threshold?      uint32
        |  +--rw suppress-threshold?   uint32
        |  +--rw suppress-time?        uint32
        |  +--rw restart-penalty?      uint32
        +--rw mtus
        |  +--rw mtu* [owner]
        |     +--rw owner    xr:Cisco-ios-xr-string
        |     +--rw mtu      uint32
        +--rw encapsulation
        |  +--rw encapsulation?         string
        |  +--rw capsulation-options?   uint32
        +--rw shutdown?                       empty
        +--rw interface-virtual?              empty
        +--rw secondary-admin-state?          Secondary-admin-state-enum
        +--rw interface-mode-non-physical?    Interface-mode-enum
        +--rw bandwidth?                      uint32
        +--rw link-status?                    empty
        +--rw description?                    string
        +--rw active                          Interface-active
        +--rw interface-name                  xr:Interface-name
```

Note: Observe that leafs have a data format like String, uint32, so on and so for; while roots does not display this information. Operations like GET and SET is dedicated to pull/update these values.

Another note is that most of the models require augments to have the full configuration, in the CLI output there is the basic interface management configuration, in case IPv4 needs to be displayed, use the next:

```
% pyang -f tree yan2/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang  yan2/vendor/cisco/xr/711/Cisco-IO
module: Cisco-IOS-XR-ifmgr-cfg
  +--rw global-interface-configuration
  |  +--rw link-status?   Link-status-enum
  +--rw interface-configurations
     +--rw interface-configuration* [active interface-name]
        +--rw dampening
        |  +--rw args?                  enumeration
        |  +--rw half-life?             uint32
        |  +--rw reuse-threshold?       uint32
        |  +--rw suppress-threshold?    uint32
        |  +--rw suppress-time?         uint32
        |  +--rw restart-penalty?       uint32
        +--rw mtus
```

```
|  +--rw mtu* [owner]
|     +--rw owner    xr:Cisco-ios-xr-string
|     +--rw mtu      uint32
+--rw encapsulation
|  +--rw encapsulation?         string
|  +--rw capsulation-options?   uint32
+--rw shutdown?                              empty
+--rw interface-virtual?                     empty
+--rw secondary-admin-state?                 Secondary-admin-state-enum
+--rw interface-mode-non-physical?           Interface-mode-enum
+--rw bandwidth?                             uint32
+--rw link-status?                           empty
+--rw description?                           string
+--rw active                                 Interface-active
+--rw interface-name                         xr:Interface-name
+--rw ipv4-io-cfg:ipv4-network
|  +--rw ipv4-io-cfg:bgp-pa
|  |  +--rw ipv4-io-cfg:input
|  |  |  +--rw ipv4-io-cfg:source-accounting?        boolean
|  |  |  +--rw ipv4-io-cfg:destination-accounting?   boolean
|  |  +--rw ipv4-io-cfg:output
|  |     +--rw ipv4-io-cfg:source-accounting?        boolean
|  |     +--rw ipv4-io-cfg:destination-accounting?   boolean
|  +--rw ipv4-io-cfg:verify
|  |  +--rw ipv4-io-cfg:reachable?      Ipv4-reachable
|  |  +--rw ipv4-io-cfg:self-ping?      Ipv4-self-ping
|  |  +--rw ipv4-io-cfg:default-ping?   Ipv4-default-ping
|  +--rw ipv4-io-cfg:bgp
|  |  +--rw ipv4-io-cfg:qppb
|  |  |  +--rw ipv4-io-cfg:input
|  |  |     +--rw ipv4-io-cfg:source?        Ipv4-interface-qppb
|  |  |     +--rw ipv4-io-cfg:destination?   Ipv4-interface-qppb
|  |  +--rw ipv4-io-cfg:flow-tag
|  |     +--rw ipv4-io-cfg:flow-tag-input
|  |        +--rw ipv4-io-cfg:source?        boolean
|  |        +--rw ipv4-io-cfg:destination?   boolean
|  +--rw ipv4-io-cfg:addresses
|  |  +--rw ipv4-io-cfg:secondaries
|  |  |  +--rw ipv4-io-cfg:secondary* [address]
|  |  |     +--rw ipv4-io-cfg:address      inet:ipv4-address-no-zone
|  |  |     +--rw ipv4-io-cfg:netmask      inet:ipv4-address-no-zone
|  |  |     +--rw ipv4-io-cfg:route-tag?   uint32
|  |  +--rw ipv4-io-cfg:primary!
|  |  |  +--rw ipv4-io-cfg:address      inet:ipv4-address-no-zone
|  |  |  +--rw ipv4-io-cfg:netmask      inet:ipv4-address-no-zone
|  |  |  +--rw ipv4-io-cfg:route-tag?   uint32
|  |  +--rw ipv4-io-cfg:unnumbered?    xr:Interface-name
|  |  +--rw ipv4-io-cfg:dhcp?          empty
|  +--rw ipv4-io-cfg:helper-addresses
|  |  +--rw ipv4-io-cfg:helper-address* [address vrf-name]
|  |     +--rw ipv4-io-cfg:address     inet:ipv4-address-no-zone
|  |     +--rw ipv4-io-cfg:vrf-name    xr:Cisco-ios-xr-string
|  +--rw ipv4-io-cfg:forwarding-enable?       empty
|  +--rw ipv4-io-cfg:icmp-mask-reply?         empty
|  +--rw ipv4-io-cfg:tcp-mss-adjust-enable?   empty
|  +--rw ipv4-io-cfg:ttl-propagate-disable?   empty
|  +--rw ipv4-io-cfg:point-to-point?          empty
|  +--rw ipv4-io-cfg:mtu?                     uint32
+--rw ipv4-io-cfg:ipv4-network-forwarding
   +--rw ipv4-io-cfg:directed-broadcast?   empty
   +--rw ipv4-io-cfg:unreachables?         empty
   +--rw ipv4-io-cfg:redirects?            empty
```

In this query two models are used: Cisco-IOS-XR-ifmgr-cfg.yang and Cisco-IOS-XR-ipv4-io-cfg.yang, and now the IPv4 address is shown as a leaf.

Note: in case you see error like: "yang/vendor/cisco/xr/711/Cisco-IOS-XR-ifmgr-cfg.yang:5: error: module "Cisco-IOS-XR-types" not found in search path" add the **--path=** in the command.

With this done and checked, any user can request information with the gNMI operations and change date, for more examples click the next link: Programmability Configuration Guide

In case the user wants to run a simple API, there are tools like: grpcc.

This API is installed via NPM, this is the tool utilized in the Programmability Configuration Guide link, said link shares more examples for users to test queries and replies.

Troubleshooting:

For gNMI it is required to check the query before collecting any entry, most of the APIs like:

- gnmic
- grpcc

- gRPC

All, display the error that the router generated.

For example:

```
"cisco-grpc:errors": {
"error": [
{
"error-type": "application",
"error-tag": "operation-failed",
"error-severity": "error",
"error-message": "'YANG framework' detected the 'fatal' condition 'Operation failed'"
}
]
}
}
```

Or

```
"error": [
   {
    "error-type": "application",
    "error-tag": "operation-failed",
    "error-severity": "error",
    "error-path": <path>,
"error-message": "'sysdb' detected the 'warning' condition 'A verifier or EDM callback function returne
   }
  ]
```

These are Platform Dependant errors that need to be checked along the router. It is suggested to check that the commands in the query can also be issued in the router via the CLI.

For this type of errors, or any other related to the Cisco IOS XR platform share the next information to TAC:

- Query that is used and operation:

```
    {
     "Cisco-IOS-XR-ifmgr-cfg:interface-configurations":
      { "interface-configuration": [
        {
          "active": "act",
          "interface-name": "Loopback0",
          "description": "LOCAL TERMINATION ADDRESS",
          "interface-virtual": [
            null
           ],
           "Cisco-IOS-XR-ipv4-io-cfg:ipv4-network": {
            "addresses": {
                "primary": {
                  "address": "172.16.255.1",
                  "netmask": "255.255.255.255"
           }
          }
        }
      }
    ]
  }
}
```

- Error that is displayed (any of the shown above).
- Issue the next command:

```
show grpc trace all
```

Please test the query a couple of times and repeat the "show grpc trace all" command.

The errors are variant but they also show the component that can generate an issue:

For example:

- "'sysdb' detected the 'warning' condition 'A verifier or EDEDM callback function returned: 'not found'": This error describe sysdb it is required to collect show tech commands for this process in the router.

  Next example shows the show tech for sysdb process showing the error.

```
show tech-support sysdb
```

  For this output, the error display a component and the error, collect any **show tech-support** that can be related to the error being displayed.

- "'YANG framework' detected the 'fatal' condition 'Operation failed'": This error does not show a process in the router, meaning that the query is failing in the model, share this information to TAC to review what can be failing.

After this information is collected, also add the next set of commands:

In XR VM:

  show tech-support tctcpsr

  show tech-support grpcc

  show tech-support gsp

  show tech-support