gRPC, gNMI, gNOI... Oh My!

An Enterprise Network Automation Journey

Jeremy Cohoe
Technical Marketing Engineer,
@jeremycohoe

Cisco Webex App

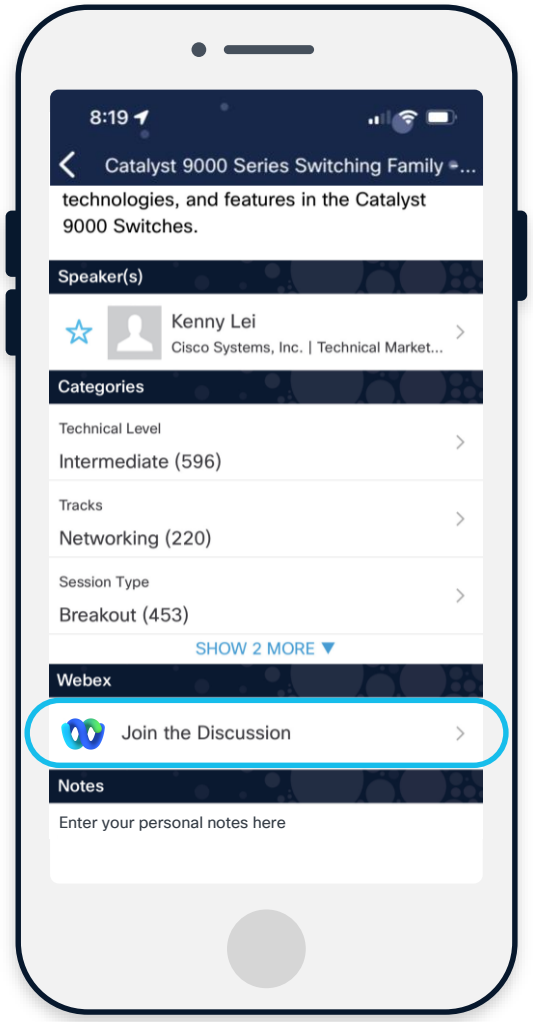
Questions?

Use Cisco Webex App to chat
with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

Webex spaces will be moderated by the speaker until June 13, 2025.



Agenda

01 Introduction

02 Zero Touch Provisioning

03 gNMI API Overview

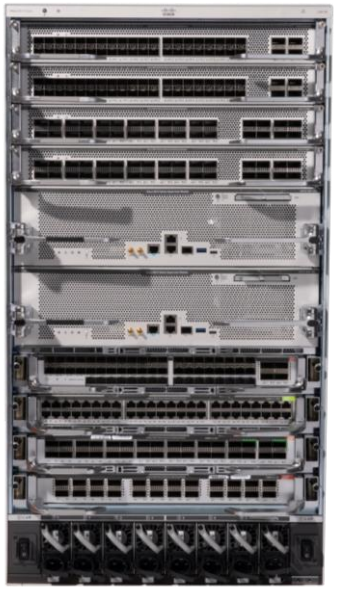
04 Telemetry

05 gNOI Workflow API’s

06 gRPC Tunnel

07 Resources & Closing

Learn more on New Cisco Smart Switches



Cisco C9610 Smart Switch

Cisco C9350 Smart Switch

Visit World of Solutions

Breakout Sessions

FULL CONFERENCE | IT LEADERSHIP

Campus Switching Innovations for Future Proofed Workspaces - BRKENS-2609

Minhaj Uddin, Leader, Technical Marketing, Cisco - **Distinguished Speaker**

Schedule **Tuesday, Jun 10 | 11:00 AM - 12:30 PM PDT | SDCC - Upper Level, Room 33A**

FULL CONFERENCE | IT LEADERSHIP

Campus Switching Architecture for Future Proofed Workspaces - BRKARC-2668

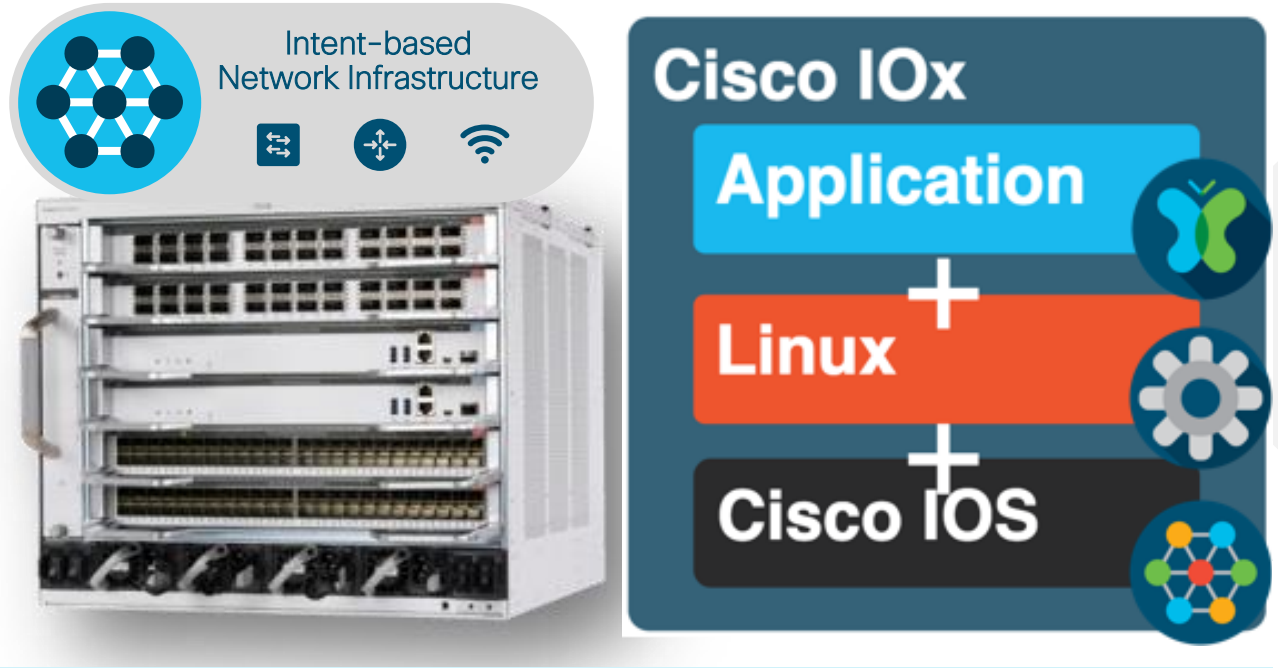
Kenny Lei, TME, Cisco - **Distinguished Speaker**

Schedule **Wednesday, Jun 11 | 3:30 PM - 5:00 PM PDT | SDCC - Upper Level, Room 28AB**

Day 0

Getting Started with Zero Touch Provisioning (ZTP)

Embedded CentOS container for IOS XE ZTP

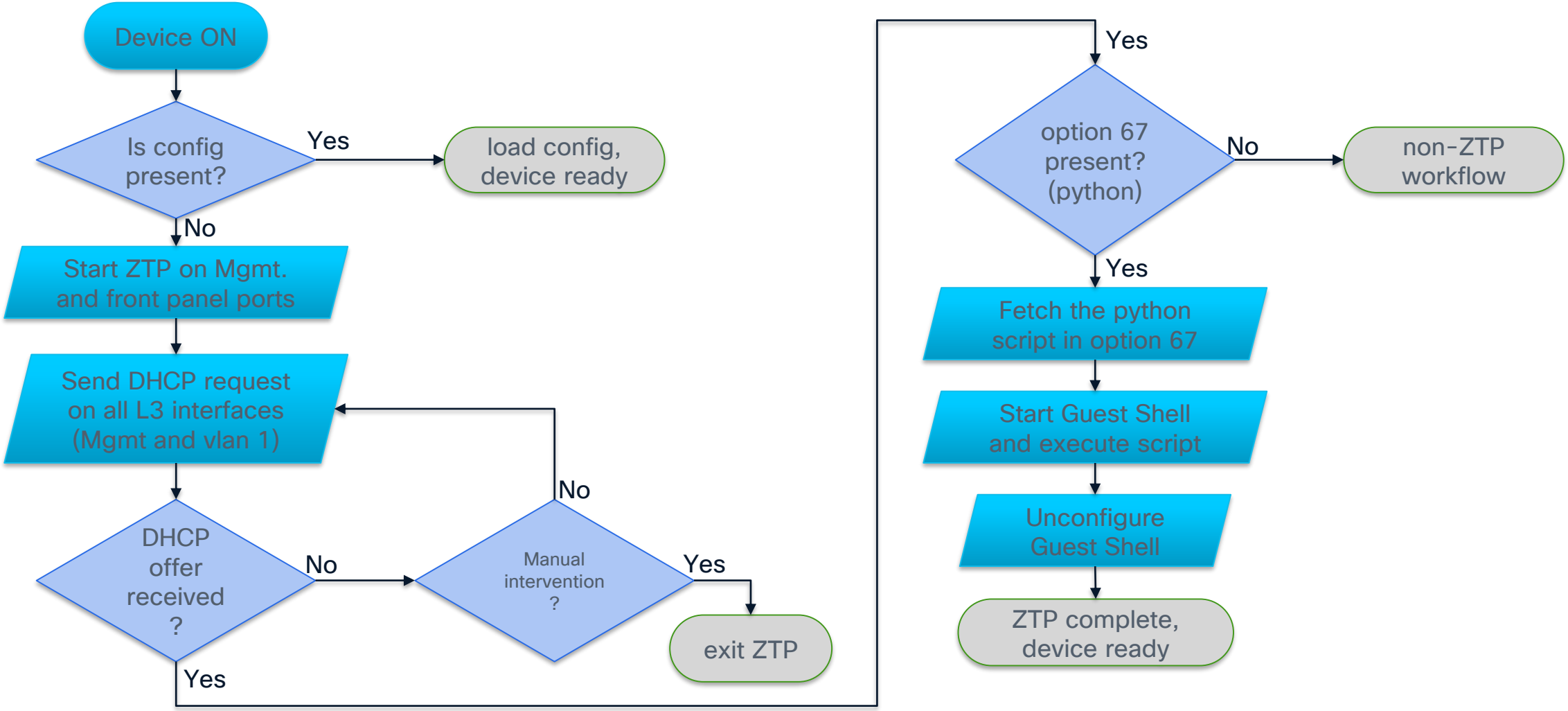


CentOS

- Fault/resource isolation
- Secure Linux shell environment
- Python interpreter, pip, Bash shell
- Python API & NETCONF API into IOS XE
- Integrated with ZTP and EEM
- Guest Shell disabled by default, enabled manually
- Enabled then disabled after ZTP completes automatically

```
iosxe# guestshell enable
iosxe# guestshell run bash
```

ZTP Workflow using Guest Shell



Python Modules - API

3 Python modules are available that are the API between Guest Shell and the IOS XE device:

- cli.cli, cli.clip
- cli.execute, cli.executep
- cli.configure, cli.configurep

```
print "\n\n *** Sample ZTP Day0 Python Script *** \n\n"
# Importing cli module
import cli

print "Configure vlan interface, gateway, aaa, and enable netconf-yang\n\n"
cli.configurep(["int vlan 1", "ip address 10.5.123.27 255.255.255.0", "no shut", "end"])
cli.configurep(["ip default-gateway 10.5.123.1", "end"])
cli.configurep(["username admin privilege 15 secret 0 XXXXXXXXXXXX"])
cli.configurep(["aaa new-model", "aaa authentication login default local", "end"])
cli.configurep(["aaa authorization exec default local", "aaa session-id common", "end"])
cli.configurep(["netconf-yang", "end"])

print "\n\n *** Executing show ip interface brief  *** \n\n"
cli_command = "sh ip int brief"
cli.executep(cli_command)

print "\n\n *** ZTP Day0 Python Script Execution Complete *** \n\n"
```

1. cli.cli(command) –This function takes an IOS command as an argument, runs the command through the IOS parser, and returns the resulting text.

2. cli.execute(command) –This function executes a single EXEC command and returns the output; however, does not print the resulting text. No semicolons or newlines are allowed as part of this command. Use a Python list with a for-loop to execute this function more than once.

3. cli.configure(command) –This function configures the device with the configuration available in commands. It returns a list of named tuples that contains the command and its result

4, 5, 6: cli.{cli, execute, configure}p (command) –This function works exactly the same as the other functions, except that it prints the resulting text to *stdout* rather than returning it .

https://developer.cisco.com/codeexchange/github/repo/jeremycohoe/IOSXE-Zero-Touch-Provisioning/

NETCONF API

The NETCONF interface on Cisco IOS XE is accessible from within the Guest Shell, which can be used at Day 0. No interface configuration or connectivity is required.

The **ncclient** Python library can be used to connect to the NETCONF interface when there is no IP connectivity, similar to the Python CLI modules and API. This can be used by ZTP at Day 0 to programmatically configure the device using either CLI or YANG.

```
C9300(config)#netconf-yang ssh port ?
  <1-65535>  Port number range (default port number is 830)
  disable    Disable external NETCONF SSH connectivity

C9300(config)#netconf-yang ssh local-vrf guestshell ?
  enable  Enable NETCONF access
  port    Configure port number for the NETCONF ssh connection
```

```
[guestshell@guestshell ~]$ ssh localhost -p 830 -l admin
admin@localhost's password:
<?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>
<capability>urn:ietf:params:netconf:base:1.1</capability>
```

ZTP

CLI API
Enable NETCONF
Enable AAA

NETCONF API
Preform RPC Actions
Programmatic Configuration of device features

```
[guestshell@guestshell ~]$ python3 ./get_hostname.py
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:20d75dd3-60b7-4b1e-9260-0438282d37c8" xmlns="urn:
        <data>
                <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
                        <hostname>C9300</hostname>
                </native>
        </data>
</rpc-reply>
```

Authentication from Guest Shell to NETCONF is still required, both credentials and certificates are supported
get_hostname.py example at https://github.com/jeremycohoe/ncclient-get-hostname
ztp-Netconf.py example at. https://github.com/jeremycohoe/IOSXE-Zero-Touch-Provisioning/blob/master/ztp-netconf.py

Enable gNMI using ZTP + python API

1. Enable GNMI API in both secure and non-secured modes
2. create certificates & allow their use
3. enable user/pass authentication
4. Create a user with privilege 1
5. Enable the read-only RBAC/NACM rules for privilege 1 users

```
2  # Enable gNMI secure API
3  cli.configurep(["gnxi", "gnxi secure-init", " gnxi secure-password-auth ",
4  "service internal", "gnxi secure-allow-self-signed-trustpoint", "end"])
5
6  # Enable gNMI non-secured API
7  cli.configurep(["gnxi", "gnxi server", "end"])
8
9  # RO username
10 cli.configurep(["username ro privilege 1 secret 0 Cisco123ro"])
11
12 # Enable API RBAC NACM for priv1 users
13 print("\n\n *** Enable API RBAC NACM for priv1 users  *** \n\n")
14 cli_command = "request platform software yang-management nacm populate-read-rules privilege 1"
15 cli.executep(cli_command)
```

```
Line 1 SUCCESS:  gnxi
Line 2 SUCCESS:  gnxi secure-init
Line 3 SUCCESS:  gnxi secure-password-auth
Line 4 SUCCESS:  service internal
Line 5 SUCCESS:  gnxi secure-allow-self-signed-trustpoint
Line 6 SUCCESS:  end

Line 1 SUCCESS:  gnxi
Line 2 SUCCESS:  gnxi server
Line 3 SUCCESS:  end

Line 1 SUCCESS:  username admin privilege 15 secret 0 Cisco123

Line 1 SUCCESS:  username ro privilege 1 secret 0 Cisco123ro

 *** Enable API RBAC NACM for priv1 users  ***


*** Saving the configuration... ***

Building configuration...
[OK]
```

ztp-simple.py Python API with ZTP

```
# Enable gNMI secure API
cli.configurep(["gnxi", "gnxi secure-init", " gnxi secure-password-auth ",
"service internal", "gnxi secure-allow-self-signed-trustpoint", "end"])

# Enable gNMI non-secured API
cli.configurep(["gnxi", "gnxi server", "end"])

# RO username
cli.configurep(["username ro privilege 1 secret 0 Cisco123ro"])

# Enable API RBAC NACM for priv1 users
print("\n\n *** Enable API RBAC NACM for priv1 users  *** \n\n")
cli_command = "request platform software yang-management nacm populate-read-rules privilege 1"
cli.executep(cli_command)
```

CLI commands

```
gnxi
gnxi secure-init
gnxi secure-password-auth
service internal
gnxi secure-allow-self-signed-trustpoint

gnxi
gnxi server

username ro privilege 1 secret 0 Cisco123ro
request platform software yang-management nacm populate-read-rules privilege
1
```

Demo time, let's run the ZTP example so the device is ready, configured, and online for the configuration management usecases next
gNMI bootstrapping described in guide at:
https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1714/b_1714_programmability_cg/m_1714_prog_gnoi_protocol.html#Cisco_Concept.dita_87c3eaca-1d91-4b9b-b2c7-22972b74aaad

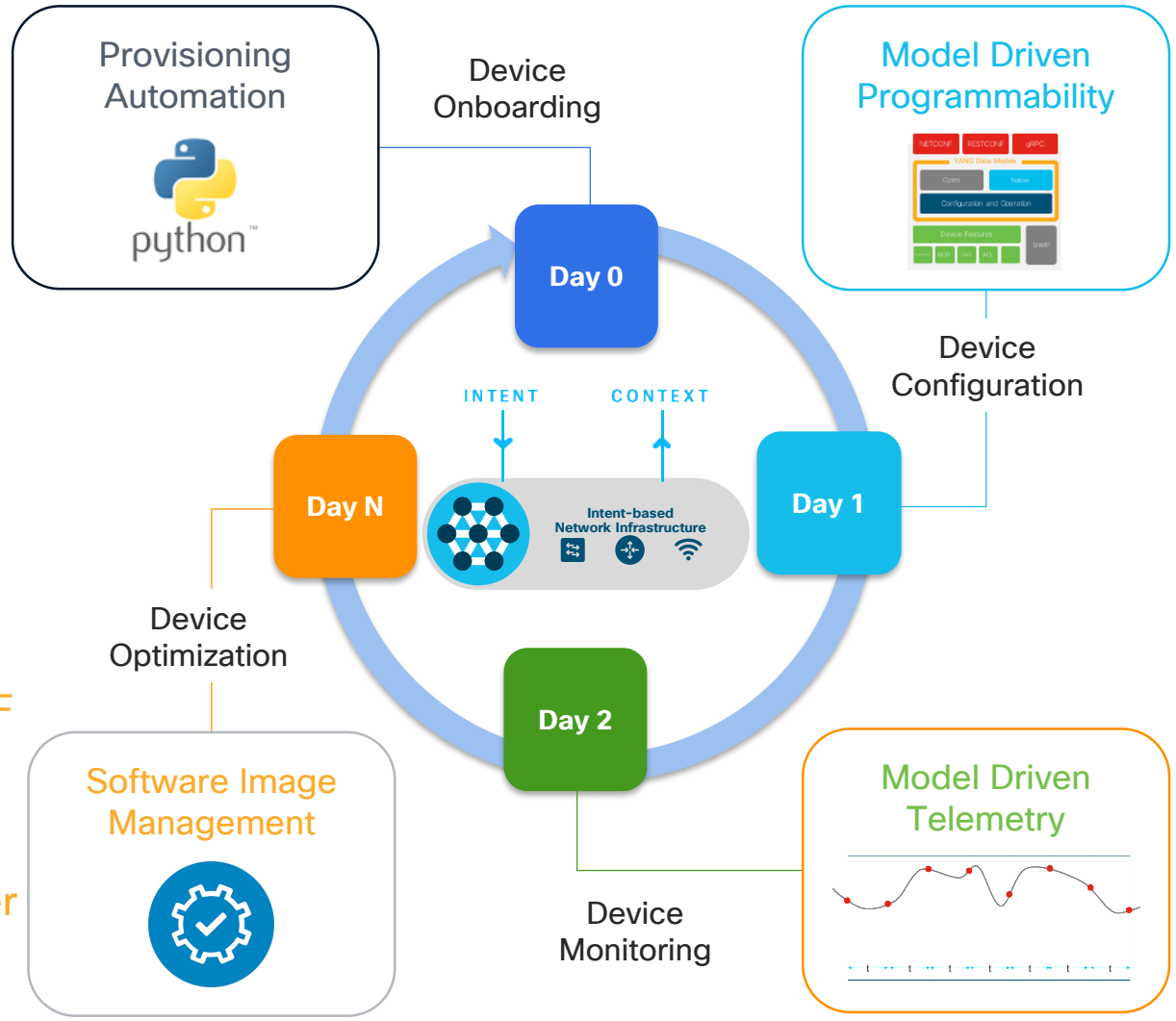
Programmability & gNMI

IOS XE Programmability & Automation Lifecycle

Pre-boot Execution Environment (iPXE)

RFC8572 Secure Zero Touch Provisioning ZTP

VM Automation



Network Configuration Protocol (NETCONF), RESTCONF, gNMI

YANG “native” Data Models, OpenConfig,

YANG Suite, Terraform, Ansible, pyATS tooling

gNOI cert/os/reset proto

Guest Shell + Python/NETCONF

CentOS 8 Python 3

Application Hosting with Docker

“show run” CLI to YANG

TIG_MDT container + guide

YANG On-Change support

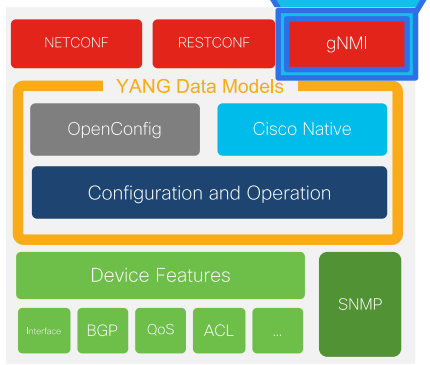
gRPC Dial-Out + DNS + mTLS

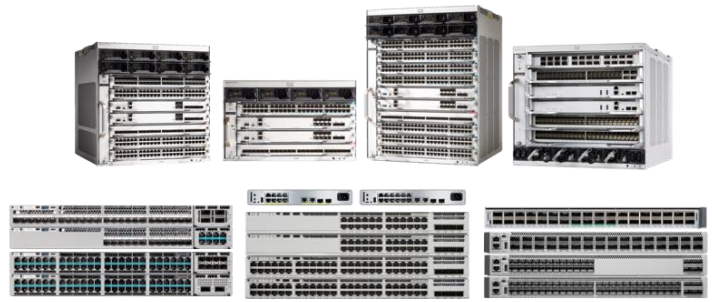
gNMI/NETCONF Dial-In

The gNMI Network Device API

NETCONF and RESTCONF (REST API) have been supported since IOS XE Release 16
Continues investment in Network API brings full gNMI API support as part of IOS XE Release 17

gNMI API





- The gNMI API supports standard operations including GET and SET for configuration management usecases
- It also supports the SUBSCRIBE operation for Telemetry
- There is a suite of micro-services for factory reset, OS image upgrade, and certificate management

gNMI is fast becoming the preferred API for both config management and telemetry usecases

Programmable Interfaces

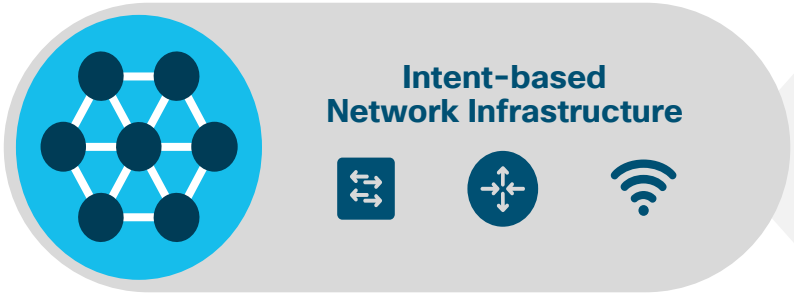
CLI

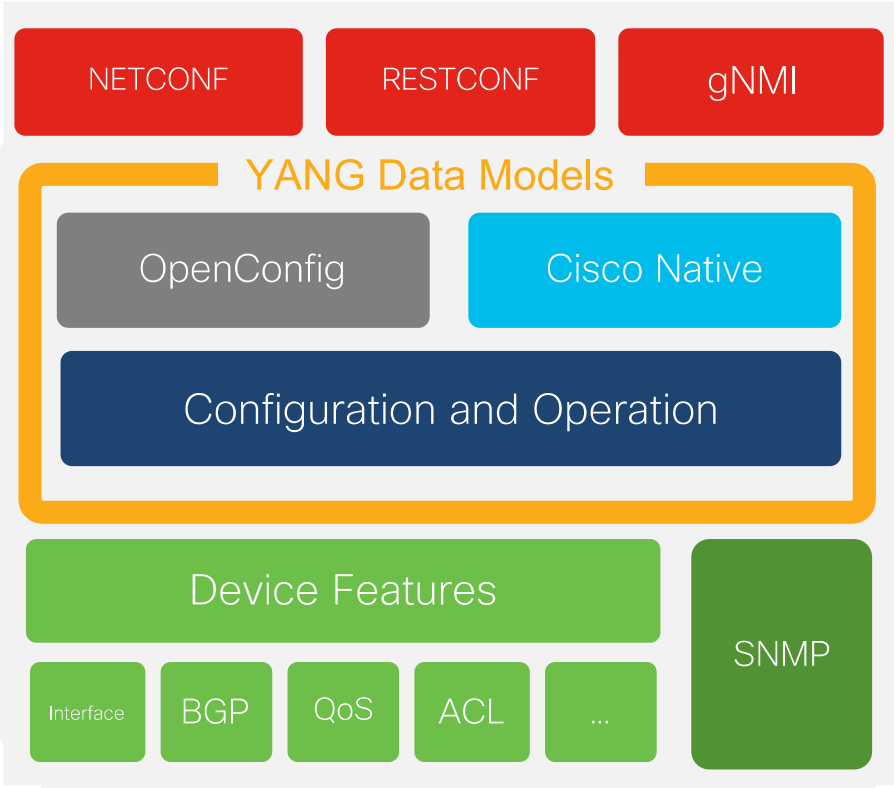
SNMP

WebUI

The NETCONF, RETCONF and gNMI are programmatic interfaces that provide **additional** methods for interfacing with the IOS XE device – Just like the CLI, SNMP, and WebUI is used for configuration changes and operational metrics so can the programmatic interfaces of NETCONF, RESTCONF and gNMI

YANG data models define the data that is available for configuration and streaming telemetry





IETF

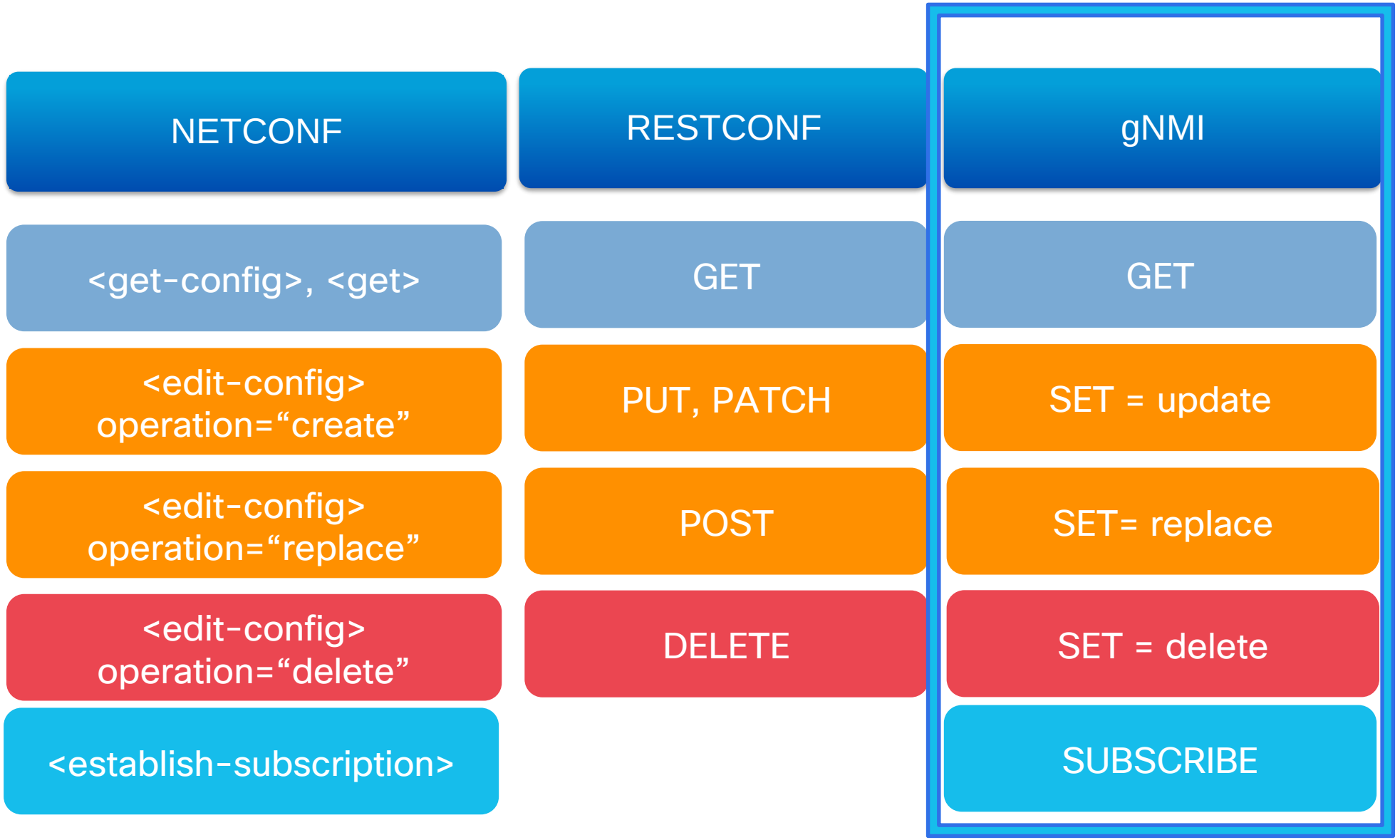
OPENCONFIG

IEEE

CISCO

cisco

API Operations

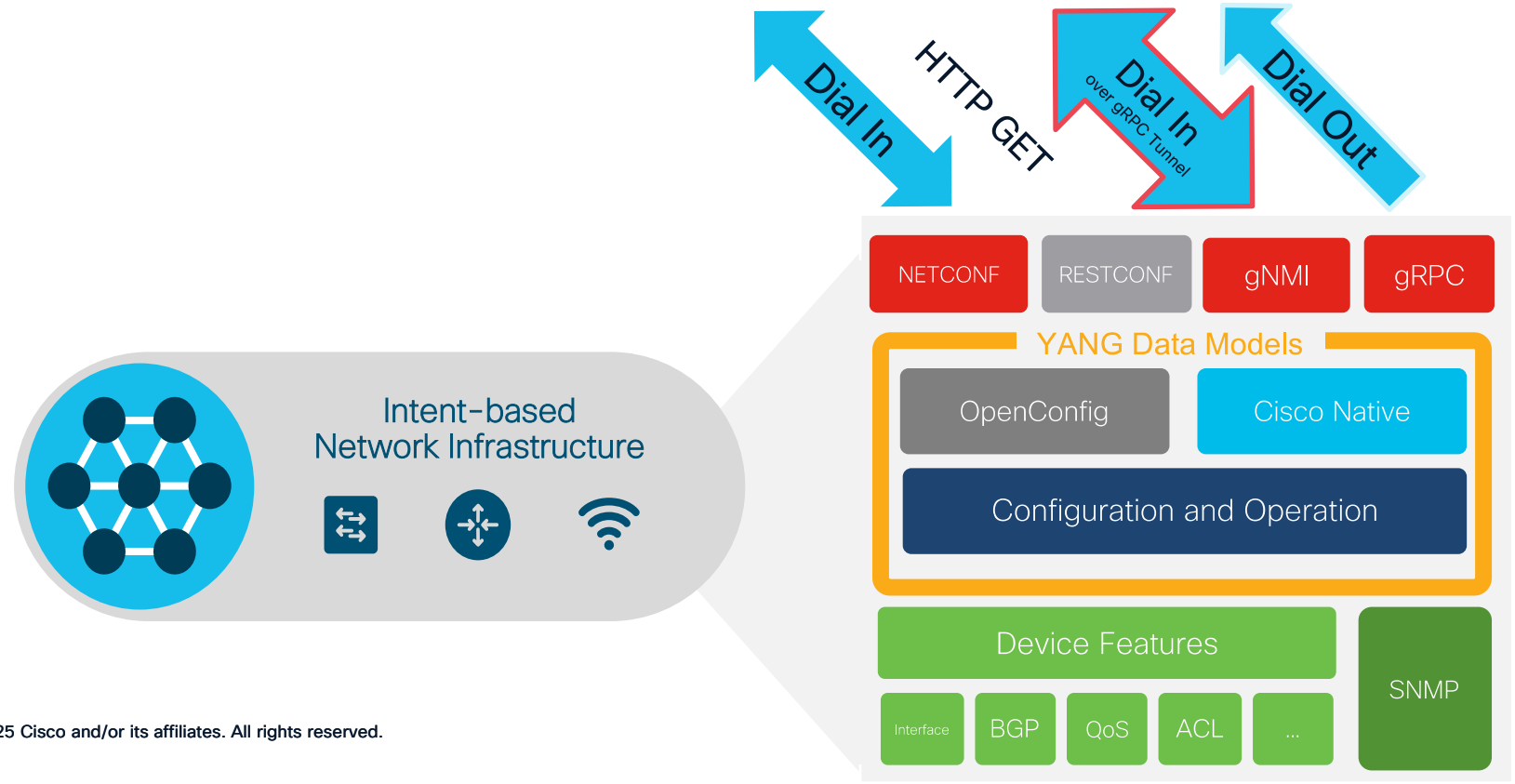


Model Driven Telemetry Interfaces

Dial In: Collector establishes a connection to the device then subscribes to telemetry (pub/sub)

Dial Out: Telemetry is pushed from the device to the collector based off configuration (push)

Publication / Subscription



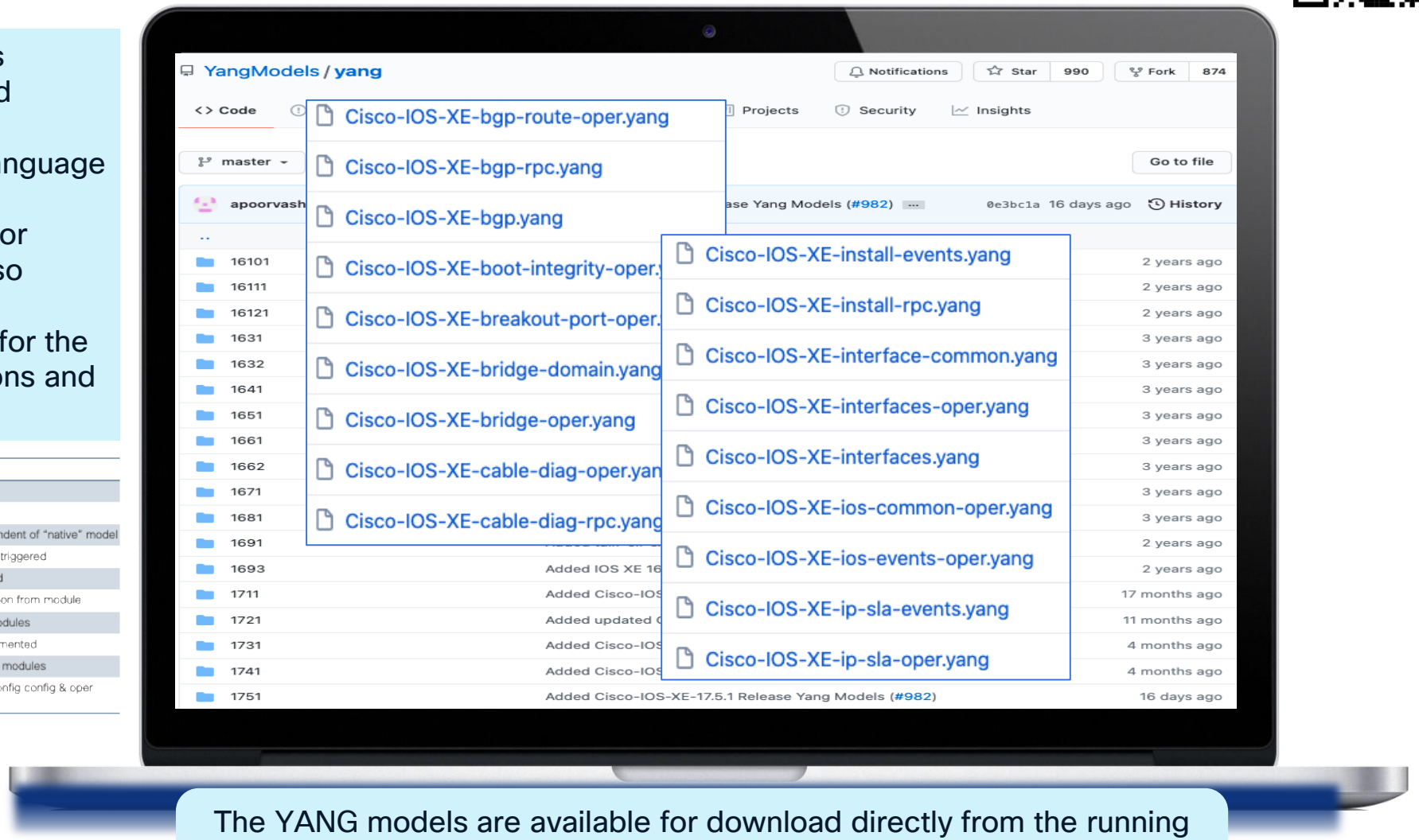
gNMI supports Dial-In Model Driven Telemetry

It also supports gRPC Tunnel where the device initiates a connection to the remote telemetry server

Cisco IOS XE – YANG model API documentation

- There are 10 types of data models including config, oper, actions, and deviations
- RFC 7950 YANG data modelling language are the API definitions for IOS XE
- The YANG modules are available for download from the API and are also published on Github.com
- Notable modules are listed below for the running-config, feature oper, actions and event notifications

	TYPE	YANG module name.yang	Description
1	Native configuration model	Cisco-IOS-XE-native	running-config
2	Operation	Cisco-IOS-XE-{feature}-oper	Feature operational data
3	Configuration	Cisco-IOS-XE-{feature}-cfg	Feature configuration, independent of "native" model
4	Events	Cisco-IOS-XE-{feature}-events	Telemetry Events that can be triggered
5	RPC	Cisco-IOS-XE-{feature}-rpc	Actions that can be performed
6	Deviation	Cisco-IOS-XE-{feature}-deviation	Device implementation deviation from module
7	Types	Cisco-IOS-XE-{feature}-types	Types – Imported by other modules
8	Obsolete	Cisco-IOS-XE-{feature}-obsolete	Obsolete should not be implemented
9	Common	Cisco-IOS-XE-{feature}-common	Common – Imported by other modules
10	Abstractions	OpenConfig-{feature} & Cisco-evpn-service	abstraction for EVPN, OpenConfig config & oper



The YANG models are available for download directly from the running IOS XE device's NETCONF, RESTCONF, or gNMI API, and from: https://github.com/YangModels/yang/tree/main/vendor/cisco/xe

gNMI

gNMI Operations

gNMI

CAP

GET

SET

SUBSCRIBE

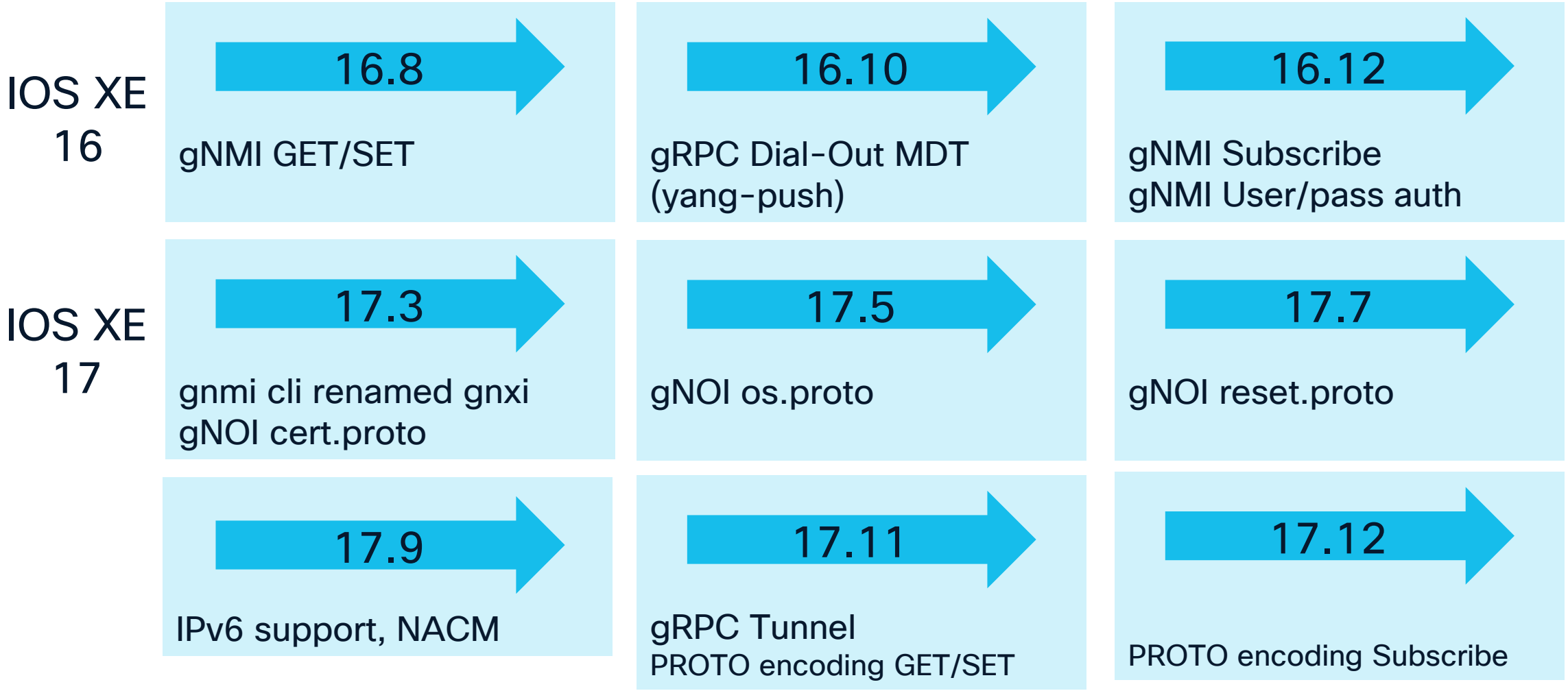
Capabilities exchange – list all supported YANG modules

Prefix	Shortcut if Paths share root paths
Paths	Defines what info is gathered
Data Type	OPER, CONFIG, ALL

Like GET	Prefix, Paths
Type	Update, Replace or Delete

Subscribe	Prefix, Path

Evolution of the gRPC API microservices

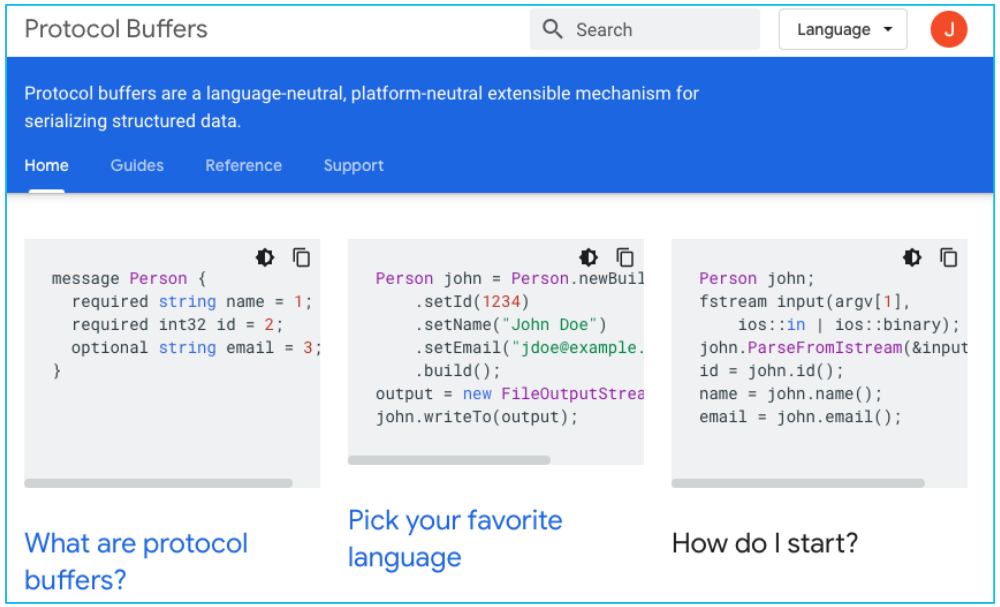


https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1713/b_1713_programmability_cg.html

Key-value pair Google Protocol Buffers (kvGPB)

A high-performance, open soxurce universal RPC framework

https://grpc.io/



Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

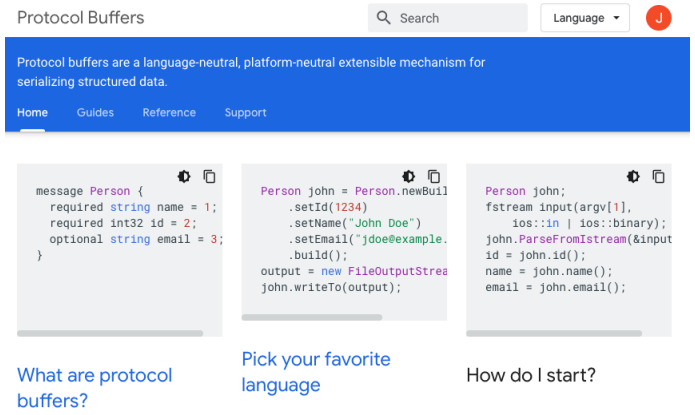
https://developers.google.com/protocol-buffers

IOS XE uses the messaging framework of gRPC to send protobuf encoded data – structured data that has been serialized into a key-value pair string

Terminology

Feature	Purpose	Release
gRPC Dial-Out	Telemetry	16.10
gRPC Dial-Out + TLS	Telemetry	17.2
gNMI Dial-In	Telemetry	16.12
gRPC	Messaging framework	X
gRPC NMI (gNMI)	Network Management	16.8
gRPC NOI (gNOI)	Network Operations	17.3
gNxI	Consolidated gRPC based microservices including gRPC, gNMI, gNOI, etc	17.3

A high-performance, open source universal RPC framework
https://grpc.io/



Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.
https://developers.google.com/protocol-buffers

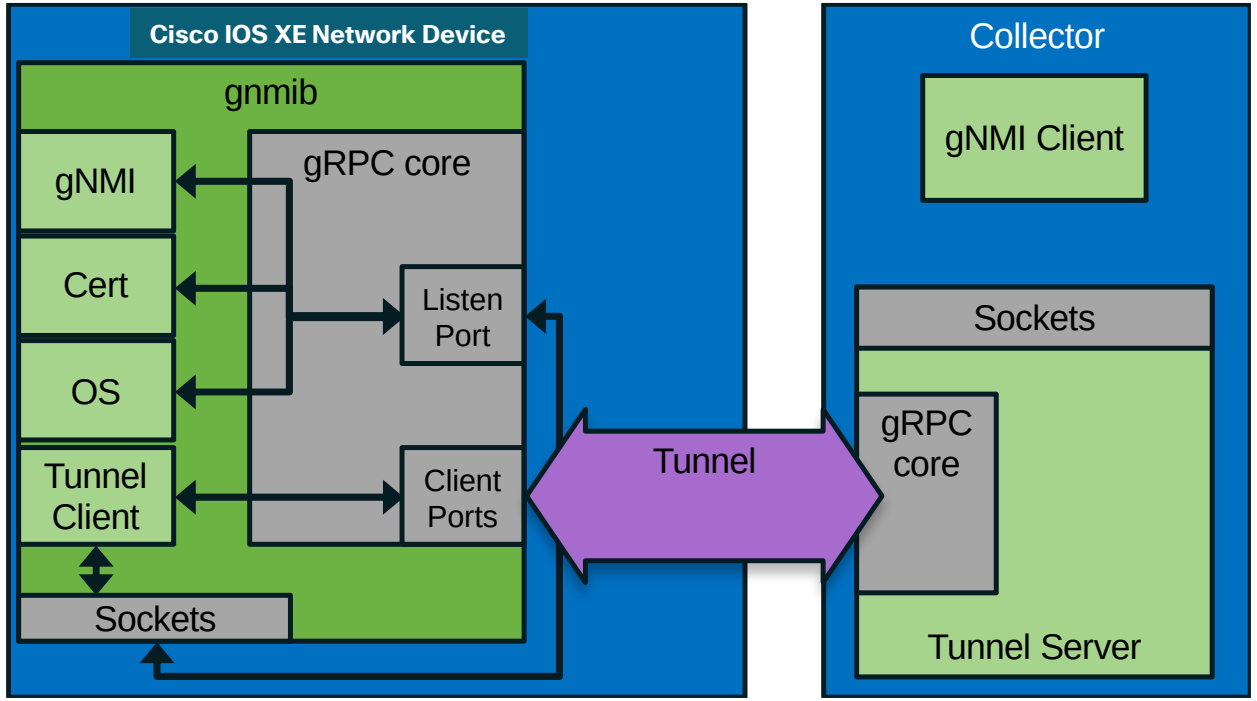
The gRPC based microservices natively use Protocol Buffers (protobuf, or .proto for short)
They also support YANG modules
config and show CLI's have changed from 'gnmi' to 'gnxi' in 17.3

gRPC tunnel

"grpctunnel is an implementation of a TCP-over-gRPC tunnel"
It is very similar to the commonly used "SSH tunnel" concept
https://github.com/openconfig/grpctunnel

- The devices makes a **secure** outbound connection to the gRPC tunnel server in order to expose the gNMI API for operational use
- Many devices can connect into a single tunnel server in order to increase operational efficiency
- Tunnels can be opened to any number of servers as needed and is not limited to a single tunnel

Useful when the network devices has a dynamic IP or is behind firewall/NAT



gNOI – gRPC Network Operations Interface

1. gRPC Network Operations Interface, or gNOI, is a set of gRPC-based microservices, used for executing operational commands on network devices
2. gNOI operations are executed against the gNMI API interface
3. gNOI is defined and implemented on a per proto basis
4. There are many protos defined - some are more mature and evolve and different pace

Protobuf RPC	Use	Related CLI	Release
Cert.proto	TLS Certificate management	crypto pki ...	17.3
Os.proto	Network Operating System management	install add file ...	17.5
Reset.proto	Factory Reset and secure wipe	write erase ...	17.7
File.proto	Not implemented on IOS XE, use YANG	copy, delete	N/A
System.proto	Not implemented on IOS XE, use YANG	reload, set boot	N/A

https://github.com/openconfig/gnoi

Show CLI and reference tooling

The show CLI can be used to understand the ports and status of the gNMI services

```
C9300#show gnxi state detail
Settings
========
  Server: Disabled
  Server port: 50052
  Secure server: Enabled
  Secure server port: 9339
  Secure client authentication: Disabled
  Secure trustpoint: gnxi-cert
  Secure client trustpoint:
  Secure password authentication: Disabled
```

```
GNMI
====
  Admin state: Enabled
  Oper status: Up
  State: Provisioned

  gRPC Server
  -----------
    Admin state: Enabled
    Oper status: Up

  Configuration service
  ---------------------
    Admin state: Enabled
    Oper status: Up

  Telemetry service
  -----------------
    Admin state: Enabled
    Oper status: Up
```

```
GNOI
====

  Cert Management service
  -----------------------
    Admin state: Enabled
    Oper status: Up
```

https://github.com/google/gnxi

gNxI Tools

- gNMI - gRPC Network Management Interface
- gNOI - gRPC Network Operations Interface

gNMI Clients:

- gNMI Capabilities
- gNMI Get
- gNMI Set
- gNMI Subscribe

gNOI Clients

- gNOI Cert
- gNOI OS
- gNOI Reset

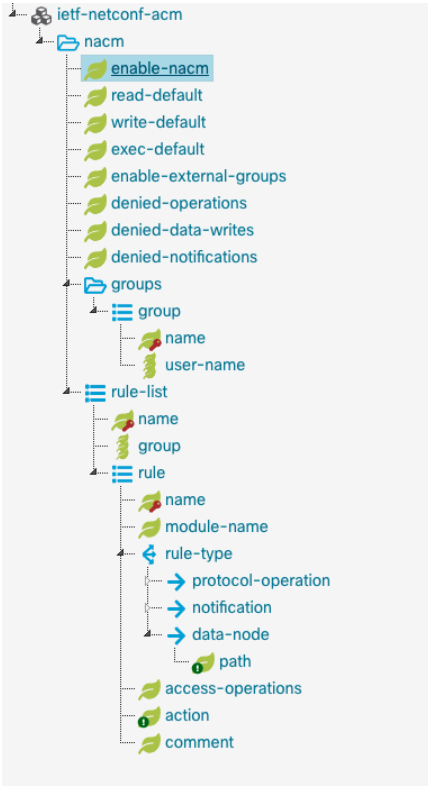
RBAC/NACM for gNMI read-only usecase

17.9.1

This feature restricts the Create, Update, Delete and Exec operations while allowing Read
GNXI must be enabled with the "**gnxi secure-password-auth**" to use NACM
CLI to populate NACM with Read Only rules:
request platform software yang-management nacm populate-read-rules privilege 1
CLI to reset NACM:
request platform software yang-management nacm reset-config

- The NACM infra was connected to gNMI in 17.9.1 to enable the access control mechanisms for gNMI users
- The NACM rules are part of the "ietf-netconf-acm" YANG data model and are not part of the running-configuration
- NACM configuration is persistent across reloads as it is part of YANG DMI
- The NACM rules can be read by performing a GET or READ operation against the NACM xpath. They can also be reset by CLI
- The "populate read rules" CLI can be used to enable read-only operations for priv1

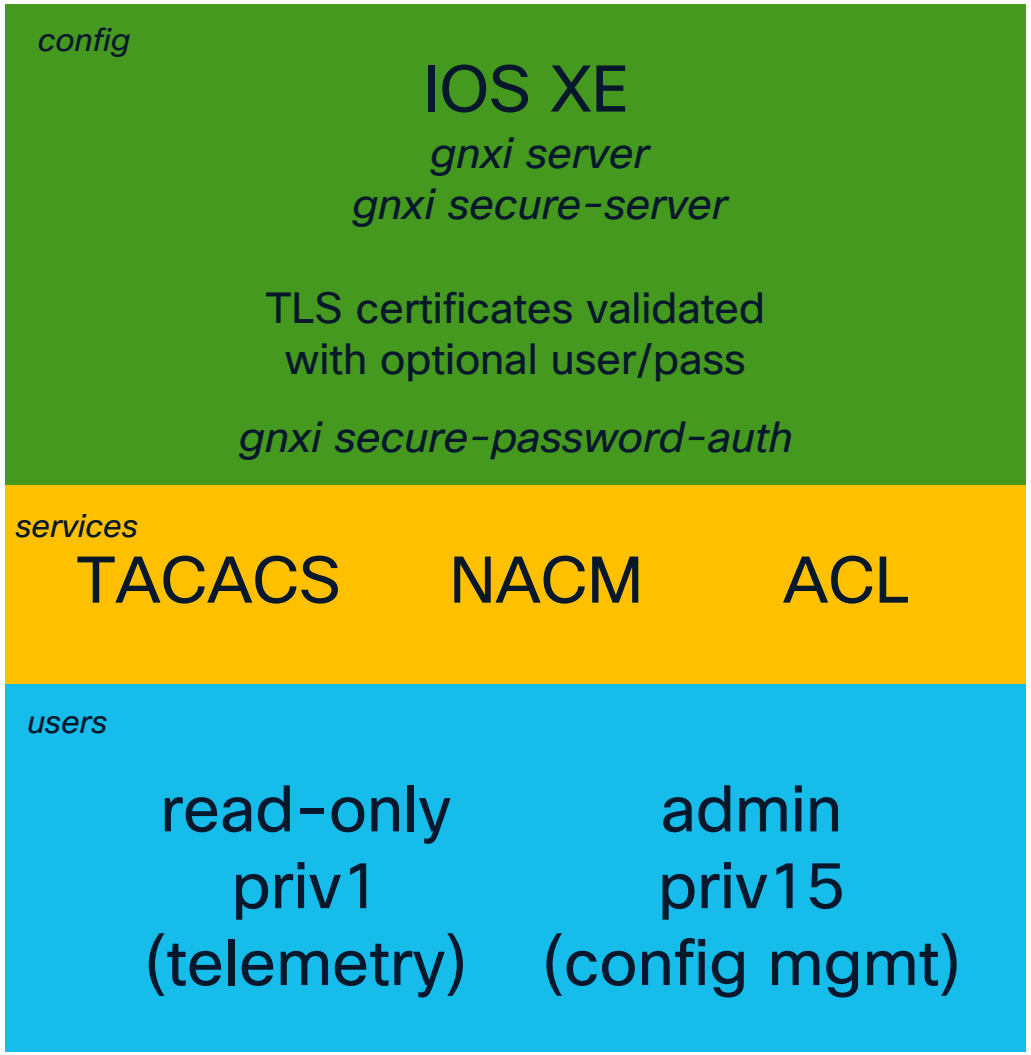
```
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
 <enable-nacm>true</enable-nacm>
 <read-default>deny</read-default>
 <write-default>deny</write-default>
 <exec-default>deny</exec-default>
 <enable-external-groups>true</enable-external-groups>
 <groups>
  <group>
   <name>gnmi-read-only-users</name>
   <user-name>gnmi_reader</user-name>
  </group>
 </groups>
 <rule-list>
  <name>admin</name>
  <group>PRIV15</group>
  <rule>
   <name>permit-all</name>
   <module-name>*</module-name>
   <access-operations>*</access-operations>
   <action>permit</action>
  </rule>
 </rule-list>
 <rule-list>
  <name>gnmi-read-only-rules</name>
  <group>gnmi-read-only-users</group>
  <rule>
   <name>deny-write-ops</name>
   <module-name>*</module-name>
   <access-operations>create update delete exec</access-operations>
   <action>deny</action>
  </rule>
  <rule>
   <name>allow-read</name>
   <module-name>*</module-name>
   <access-operations>read</access-operations>
   <action>permit</action>
  </rule>
 </rule-list>
</nacm>
```



gNMI Best Practices

Considerations when using GNXI in production

1. Authentication: Is a TLS cert used for auth, or a user/pass combo ?
2. Login Authorization with upstream TACACS/RADIUS/ISE or local ?
3. Model Based Authorization with NACM policy ?
4. Does the gNMI service need protection with an IP ACL ?
5. Low priv read-only user can only subscribe to MDT while admin user can manage the device configuration



gNMI CLI’s

```
service internal
!required if using selfsigned-trustpoint – not recommended!
gnxi
gnxi secure-allow-self-signed-trustpoint
gnxi secure-password-auth
gnxi secure-init
! secure-init once only to add below trustpoint
gnxi secure-trustpoint TP-self-signed-2328416372
gnxi secure-server
! read–only and disable gNOI
gnxi read-only
no gnxi enable-gnoi
!gnxi server ! keep the non-secure server disabled
```

gNXI read-only

17.15
17.12.4
17.9.6

This enhancement adds support for a read-only use-case blocking any 'write' RPCs including gNMI Set and gNOI RPC's

Disable gNOI microservices

- Block **all gNOI RPC's** including for certificate management, OS image, and factory reset services.
- Any gNOI RPCs will return a gRPC UNIMPLEMENTED error if called

switch(config) # no gnxi enable-gnoi

Set gNMI API to read-Only

- Disable any gNMI **Set** operations at the gNMI broker level
- Independent of NACM policy this disables SET entirely

switch(config)# gnxi read-only

```
VNC2-LEAF1(config)#gnxi read-only ?
  <cr>  <cr>

VNC2-LEAF1(config)#gnxi enable-gnoi ?
  <cr>  <cr>
```

```
Apr 17 22:44:26.288: %HA_EM-6-LOG: catchall: configure terminal
Apr 17 22:45:11.094: %HA_EM-6-LOG: catchall: no gnxi enable-gnoi
Apr 17 22:45:11.096: %GNMIB-5-SRV_OPER_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] operational status: DOWN
Apr 17 22:45:11.099: %GNMIB-5-SRV_ADMIN_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] administrative state: DISABLED
Apr 17 22:45:11.857: %GNMIB-5-SRV_ADMIN_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] administrative state: ENABLED
Apr 17 22:45:21.859: %GNMIB-5-SRV_OPER_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] operational status: UP
```

```
VNC2-LEAF1(config)#gnxi read-only
VNC2-LEAF1(config)#
Apr 17 22:47:34.297: %HA_EM-6-LOG: catchall: gnxi read-only
Apr 17 22:47:34.298: %GNMIB-5-SRV_OPER_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] operational status: DOWN
Apr 17 22:47:34.301: %GNMIB-5-SRV_ADMIN_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] administrative state: DISABLED
Apr 17 22:47:35.046: %GNMIB-5-SRV_ADMIN_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] administrative state: ENABLED
Apr 17 22:47:45.049: %GNMIB-5-SRV_OPER_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] operational status: UP
```

```
gNMI SET
========
update {
  path {
    origin: "rfc7951"
    elem {
      name: "Cisco-IOS-XE-native:native"
    }
    elem {
      name: "hostname"
    }
  }
  val {
    json_ietf_val: "\"tried-to-set-with-gnmi\""
  }
}
```

```
ERROR: gNMI/Set not available in read-only mode
```

Increased Logging for gNMI

17.6
17.7

17.6: GNXI OS history CLI
show history of GNXI OS RPC's
switch# show gnxi os internal history

17.7: More granular logging introduced
Set debug level with 'set platform software trace' CLI

switch# set platform software **trace gnmi** switch active r0 ?

(...)	
gnmi	GNMI Library
gnmib-cert	Cert.proto workflow
gnmib-grpc	GRPC core
gnmib-infra	GNMI Broker Infra
gnmib-os	Os.proto workflow
gnmib-reset	Reset.proto workflow
gnxi	GNXI Library

17.7: GNMI & GNOI maintain counters per RPC/message
switch# show gnxi state stats

```
cat9300x-pod05a#show gnxi os internal histor
Truncated history of install infra events
Date/Time         UUID                                 Install Status               Install action    Install substate
-----------------------------------------------------------------------------------------------------------------------
04/17/24 11:37:56  3b77bc55-4a4e-4143-a759-bb9613013f3b  Install Start                Remove            Operation Invoked
04/17/24 11:37:56  3b77bc55-4a4e-4143-a759-bb9613013f3b  Install Complete             Remove            None

cat9300x-pod05a#

jcohoe-c9300#set platform software trace gnmi switch active r0 gnmi ?
  debug      Debug messages
  emergency  Emergency possible message
  error      Error messages
  info       Informational messages
  noise      Maximum possible message
  notice     Notice messages
  verbose    Verbose debug messages
  warning    Warning messages
```

```
jcohoe-c9300#sh gnxi state stats
GNMI
====
  Get: 4
  Set: 0
  Capabilities: 7
  Subscribe: 0

GNOI CERT
=========
  Get: 0
  Install: 0
  Rotate: 0
  Revoke: 0
  Cert CSR: 0

GNOI OS
=======
  Install: 1
  Activate: 0
  Verify: 0
```

Syslogs for gNMI

17.13

Implement syslogs for gNMI RPC's and states, gNOI gRPC's, and outgoing gRPC Tunnel connections

1. GNMI RPC received and completed GET, SET, Subscribe
2. Telemetry Dialout connection to receiver
3. Telemetry Dialout disconnection from receiver
4. GNMI gRPC Tunnel connection to server
5. GNMI gRPC Tunnel target registration
6. gNOI Certificate Management, OS Install

gNMI Get/Set

```
May  9 20:45:12.485: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Connection from ipv4:5.251.17.87:39656 for RPC gNMI/Get
May  9 20:45:12.525: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Disconnection from ipv4:5.251.17.87:39656 for RPC gNMI/Get

May  9 20:45:42.536: %SYS-5-CONFIG_P: Configured programmatically by process iosp_dmiauthd_conn_100001_vty_100001 from console as admin on vty63
May  9 20:45:42.399: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Connection from ipv4:5.251.17.87:39664 for RPC gNMI/Set
May  9 20:45:42.532: %DMI-5-CONFIG_I: Switch 1 R0/0: dmiauthd: Configured from NETCONF/RESTCONF by admin, transaction-id 1886
May  9 20:45:42.560: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Disconnection from ipv4:5.251.17.87:39664 for RPC gNMI/Set
```

gNMI Subscribe and gRPC Tunnel

```
May 10 13:00:25.320 EDT: %GNMIB-6-GRPC_TUNN_STATE: Switch 1 R0/0: gnmib: Connected to gRPC Destination ott-ads-365 (161.44.237.133) with Targets[GNMI_GNOI] using 0.0.0.0:Mgmt-vrf
May 10 13:00:25.323 EDT: %GNMIB-6-GRPC_TUNN_SESS: Switch 1 R0/0: gnmib: Received gRPC Tunnel session request from ott-ads-365 for Target Type GNMI_GNOI
May 10 13:00:25.389 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Connection (Tunneled from ott-ads-365) from ipv4:127.0.0.1:55066 for RPC gNMI/Subscribe

May 10 13:00:35.410 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Disconnection from ipv4:127.0.0.1:55066 for RPC gNMI/Subscribe
May 10 13:00:35.410 EDT: %MDT_CONNECTION-6-STATE_CHANGED: Switch 1 R0/0: pubd: Connection state changed (id 26 - 127.0.0.1:55066:0:0.0.0.0): DOWN
```

gNOI Cert operations

```
May 10 12:43:55.961 EDT: %CRYPTO_ENGINE-5-KEY_DELETED: A key named pygnoiclienttest has been removed from key storage
May 10 12:43:55.961 EDT: %PKI-6-TRUSTPOINT_DELETE: Trustpoint: pygnoiclienttest deleted succesfully
May 10 12:43:56.049 EDT: %PKI-6-TRUSTPOINT_CREATE: Trustpoint: pygnoiclienttest created succesfully
May 10 12:43:55.960 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Connection from ipv4:5.251.17.87:49478 for RPC Cert/Revoke
May 10 12:43:55.966 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Disconnection from ipv4:5.251.17.87:49478 for RPC Cert/Revoke
May 10 12:43:56.047 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Connection from ipv4:5.251.17.87:49478 for RPC Cert/Install
May 10 12:43:56.438 EDT: %CRYPTO_ENGINE-5-KEY_ADDITION: A key named pygnoiclienttest has been generated or imported by crypto-engine
```

gNOI OS

```
May 10 12:52:17.757 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Connection from ipv4:5.251.17.87:49546 for RPC OS/Install
May 10 12:52:17.764 EDT: %INSTALL-5-INSTALL_COMPLETED_INFO: Switch 1 R0/0: install_mgr: Completed install remove
May 10 12:52:17.829 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Disconnection from ipv4:5.251.17.87:49546 for RPC OS/Install
May 10 12:52:24.099 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Connection from ipv4:5.251.17.87:49550 for RPC OS/Activate
May 10 12:52:24.100 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Disconnection from ipv4:5.251.17.87:49550 for RPC OS/Activate
May 10 12:52:31.266 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Connection from ipv4:5.251.17.87:49552 for RPC OS/Verify
May 10 12:52:31.267 EDT: %GNMIB-6-GRPC_CON_STATE: Switch 1 R0/0: gnmib: Received gRPC Disconnection from ipv4:5.251.17.87:49552 for RPC OS/Verify
```

Show tech enhancements

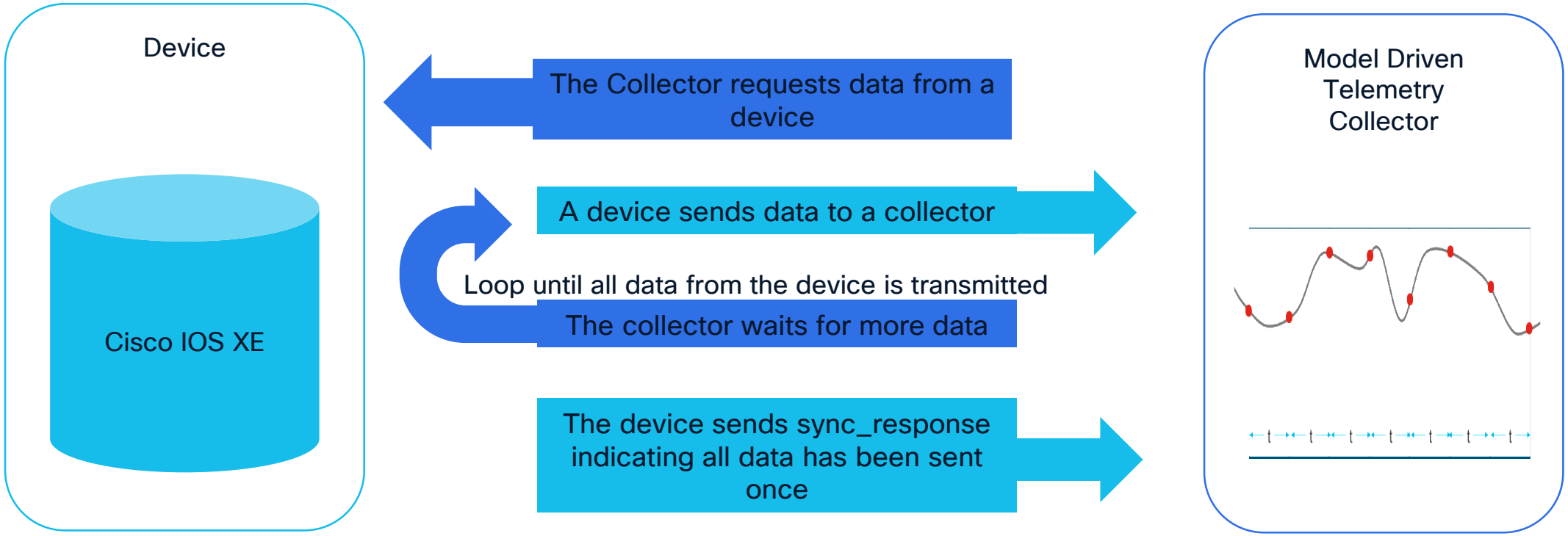
17.16

Now when running “show tech” CLI the following gNMI details are included

show tech-support yang-management
show crypto pki certificates
show crypto pki trustpoints
show gnxi state
show gnxi state detail
show gnxi state stats
show gnxi os activate
show gnxi os detail
show gnxi os install
show gnxi os internal history
show gnxi os summary
show install summary
show install active
show install committed
show install inactive
show gnxi grpctunnel destinations

gNMI sync_response

This provides an indication to the client that all of the existing data for the subscription has been sent at least once

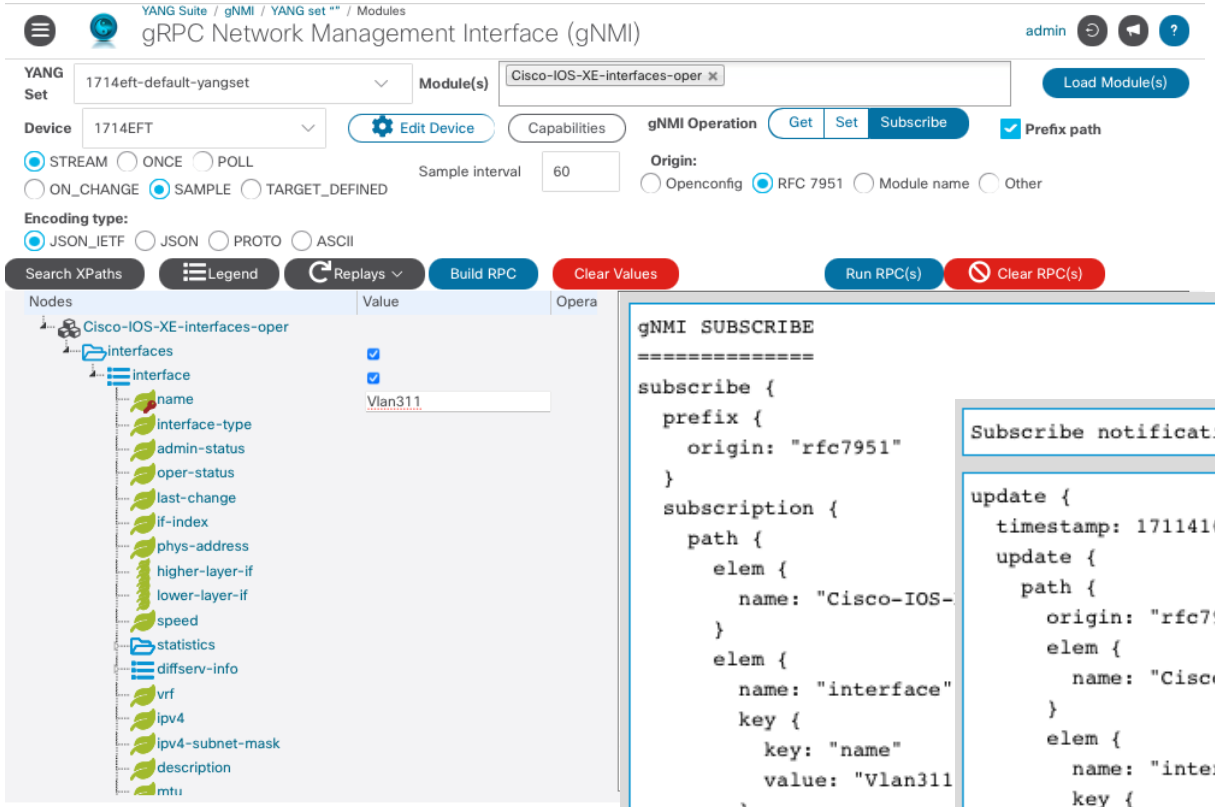


Specification defined in the OpenConfig gNMI specification on Github:
https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md

gNMI sync_response example

17.14

1. Build Subscribe Request
2. Send the Request
3. Receive & decode the Response
4. sync_response is sent from device to collector




```
gNMI SUBSCRIBE
==============
subscribe {
  prefix {
    origin: "rfc7951"
  }
  subscription {
    path {
      elem {
        name: "Cisco-IOS-
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "Vlan311
        }
      }
    }
    mode: SAMPLE
    sample_interval: 6000
  }
  encoding: JSON_IETF
}
```

```
Subscribe notification active
```

```
update {
  timestamp: 1711410791139510000
  update {
    path {
      origin: "rfc7951"
      elem {
        name: "Cisco-IOS-XE-interfaces-
      }
      elem {
        name: "interface"
        key {
          key: "name"
          value: "Vlan311"
        }
      }
    }
    val {
      json_ietf_val: "{\"name\":\"Vlan3
    }
  }
}
```

```
  "ether-stats": {
    "in-mac-control-frames": "0",
    "in-mac-pause-frames": "0",
    "in-oversize-frames": "0",
    "in-jabber-frames": "0",
    "in-fragment-frames": "0",
    "in-8021q-frames": "0",
    "out-mac-control-frames": "0",
    "out-mac-pause-frames": "0",
    "out-8021q-frames": "0"
  }
}
```

```
Xpath/Value
===========
[{'datatype': 'key',
  'value': 'Vlan311',
  'xpath': '/Cisco-IOS-XE-interfaces-oper:interfaces/interface/name'}]
```

```
update {
  timestamp: 1711410791141977000
}
```

```
Processing returns...
```

```
'update'
```

```
Subscribe sync_response
```

```
Waiting for notification
```

```
Waiting for notification
```

PROTO encoding for gNMI subscribe

17.11

In addition to JSON_IETF encoding, there is also support for PROTO encoding for gNMI periodic subscriptions.

The PROTO encoding mechanism uses the binary encoding format for both path and value to increase the efficiency of telemetry data transfer. With JSON_IETF the aggregated data is sent to the collector and with proto there is more granularity in the transmitted data.

Both Telegraf and YANG Suite already support PROTO

```
120 lines (106 sloc)  2.69 KB
1   # telegraf-gnmi-proto.conf
2   [[inputs.gnmi]]
3   redial = "60s"
4   #encoding = "json_ietf"
5   encoding = "proto"
6   addresses = ["10.85.134.70:50052"]
7   username = "admin"
8   password = ""
```

```
gNMI SUBSCRIBE
==============
subscribe {
  subscription {
    path {
      origin: "openconfig"
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
      }
    }
    mode: SAMPLE
    sample_interval: 30000000000
  }
  encoding: PROTO
}
```

The “proto” encoding restriction will be removed from the guide as it will become supported in release 17.11

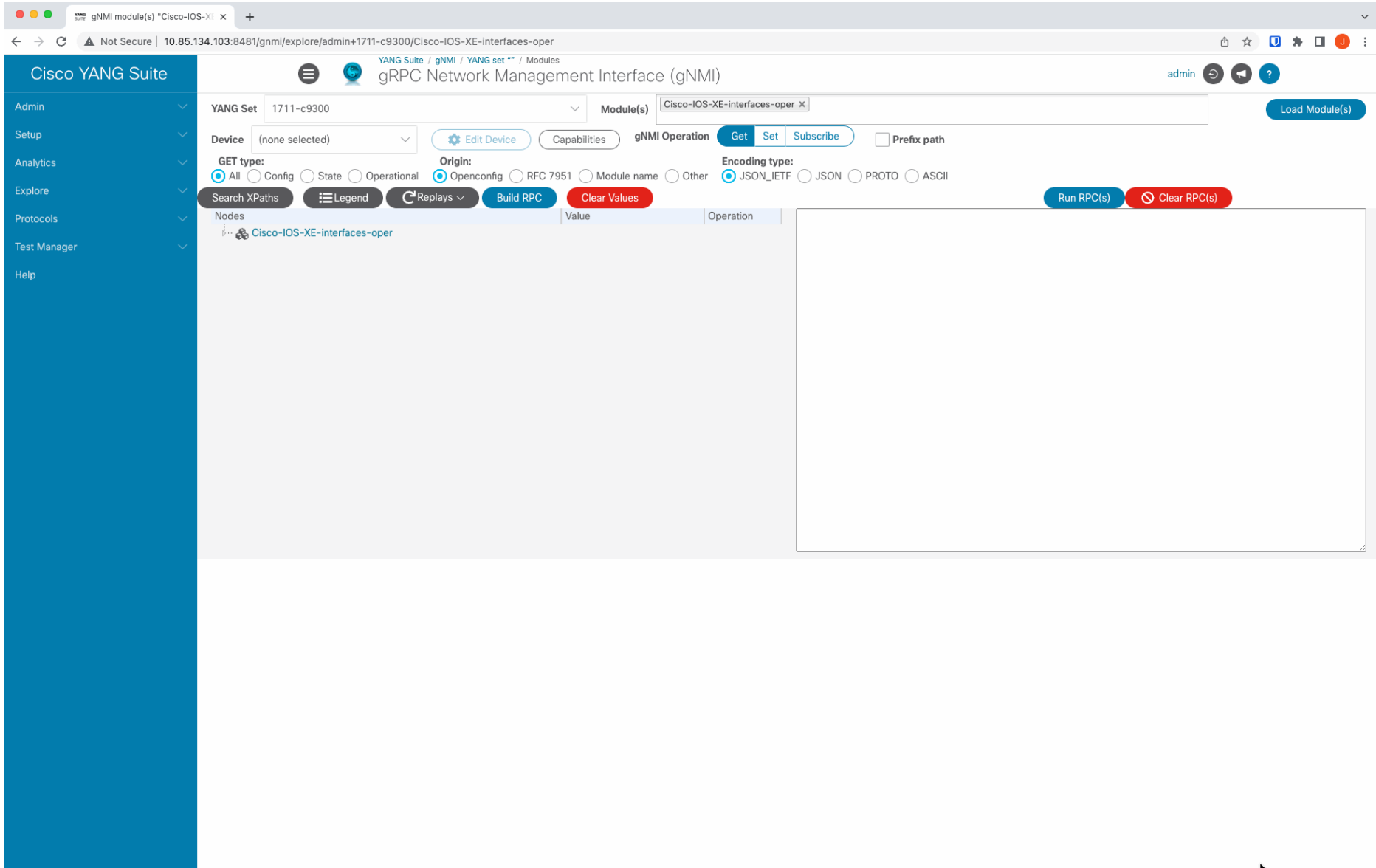
Restrictions for the gNMI Protocol

The following restrictions apply to the feature:

- JSON, BYTES, PROTO, and ASCI encoding options are not supported.

https://github.com/jeremycohoe/cisco-ios-xe-mdt/blob/master/telegraf-gnmi-proto.conf#L5
https://github.com/openconfig/gnmi/blob/master/proto/gnmi/gnmi.proto
https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/179/b_179_programmability_cg/m_178_prog_gnmi.html

gNMI subscribe PROTO demo



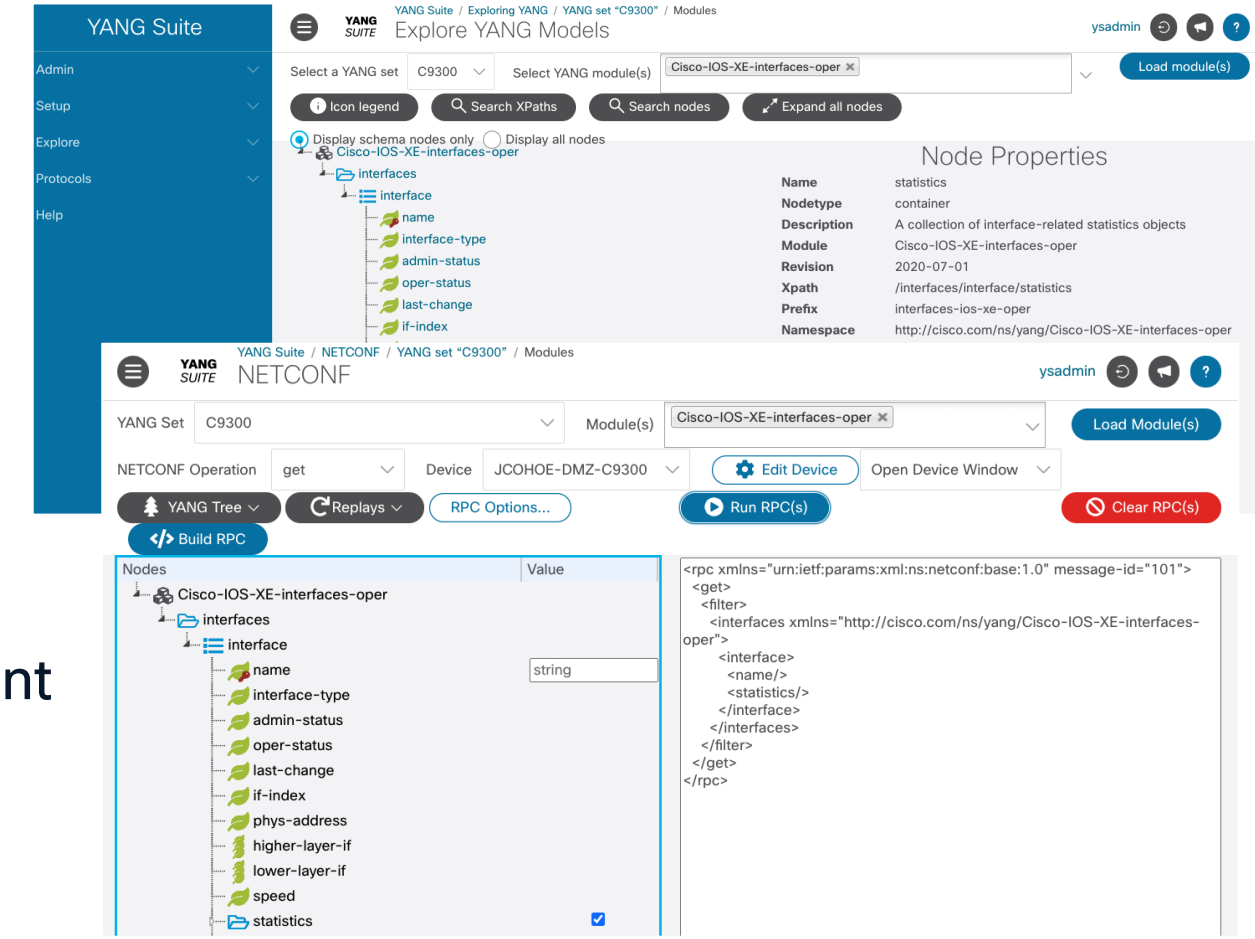
YANG Suite + gNMI

Cisco YANG Suite

YANG API Testing and Validation Environment

Construct and test YANG based APIs over NETCONF, RESTCONF, gRPC and gNMI

IOS XE / IOS XR / NX OS platforms



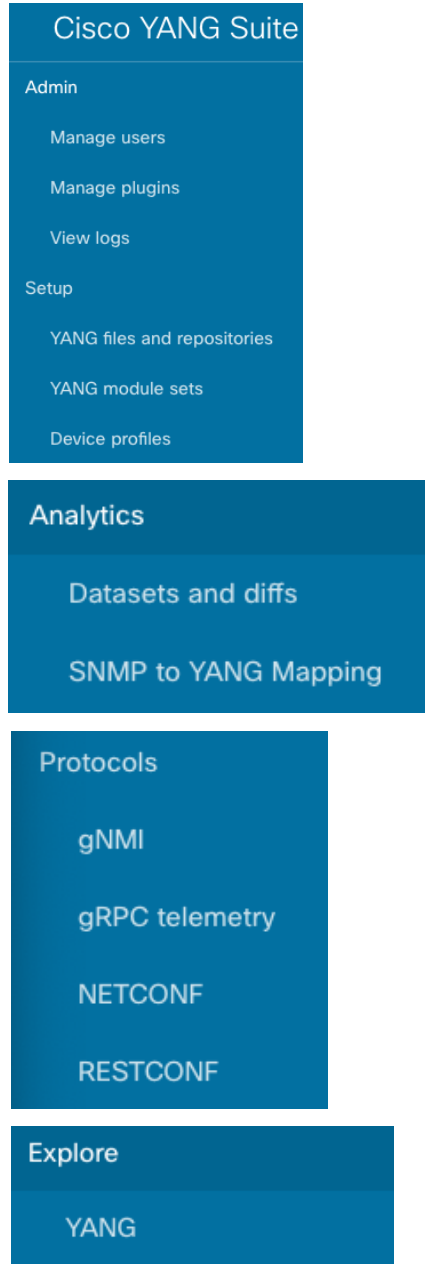
developer.cisco.com/yangsuite

Get hands-on using the new learning lab!
https://developer.cisco.com/learning/labs/intro-yangsuite/
Docker container innovation 1 container

github.com/CiscoDevNet/yangsuite

YANG Suite now includes ...

- Initial Release:
  - Plugin and YANG File Manager, Datasets and diffs
  - Device Manager
  - NETCONF (Python), gRPC Telemetry
  - Docker install support with HTTPS
- Second Release:
  - RESTCONF
  - gNMI
  - Python Integrations
- Third Release:
  - gRPC Telemetry with TLS Support
  - SNMP OID to YANG Xpath Mapping
  - Ansible Integrations
  - Pip install support



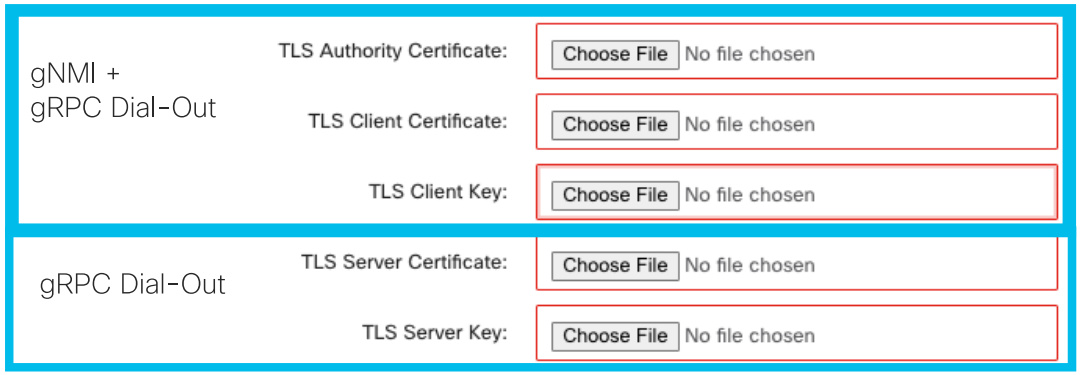
YANG Suite gNMI plugin

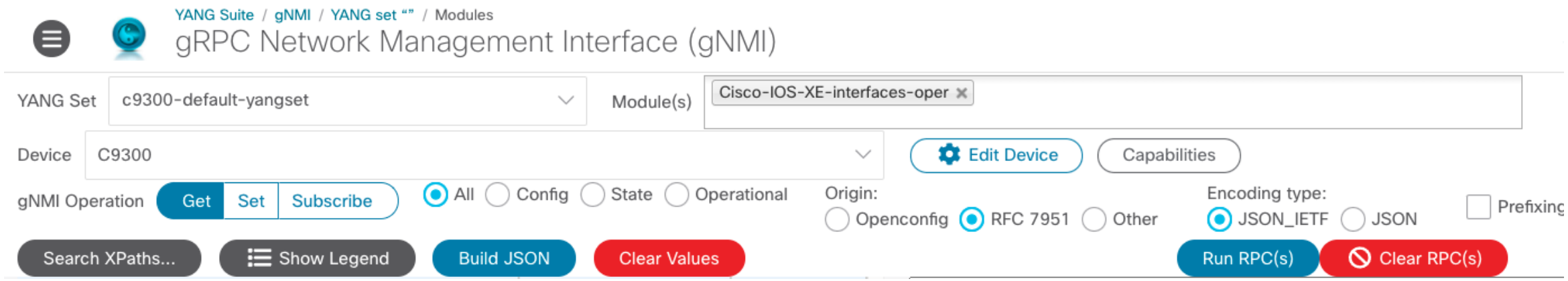
- Model-driven configuration and retrieval of config and operational data using the gRPC Network Management Interface (gNMI)
- Capabilities, Get, Set and Subscribe remote procedure calls (RPCs) supported with JSON, JSON_IETF and Proto encoding
- This fully functional gNMI client helps build, test, and validate gNMI YANG payloads

Demo's & Examples

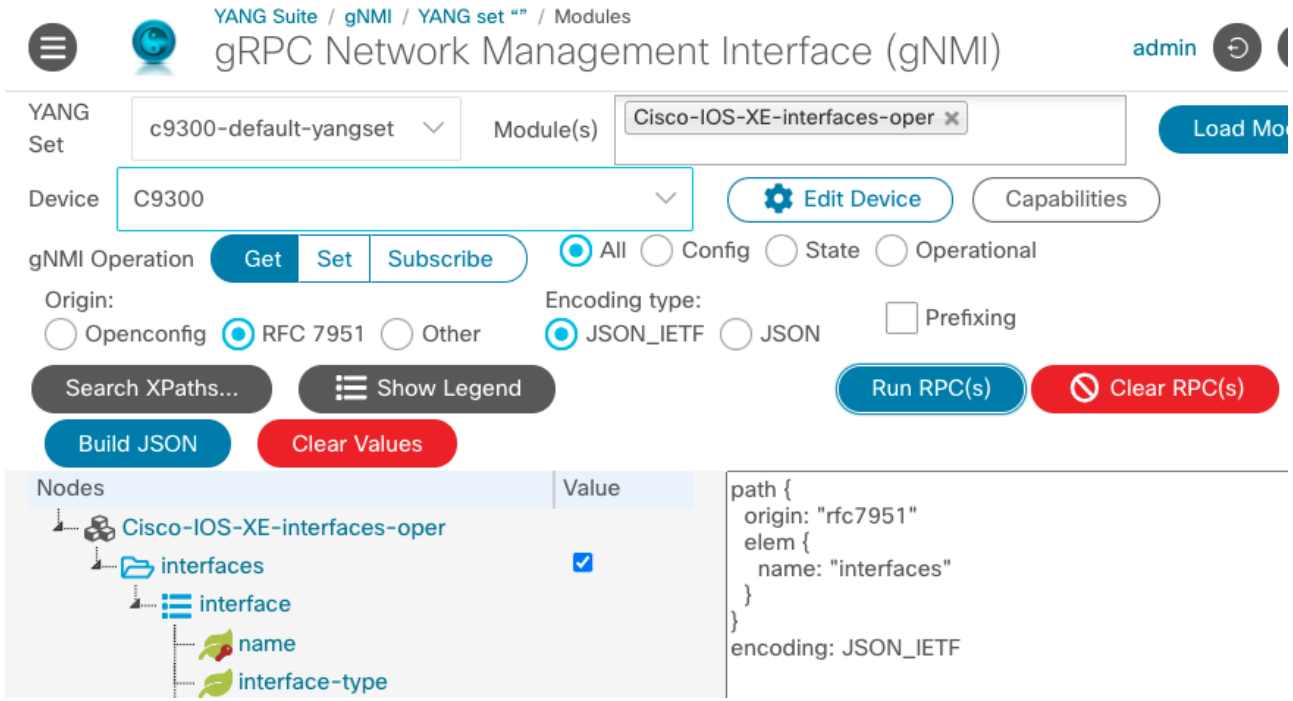
1. gNMI GET for Cisco Native hostname
2. gNMI SET for Cisco Native hostname
3. gNMI GET OC Hostname
4. gNMI SET OC Hostname
5. gNMI GET IETF Interfaces
6. gNMI Subscribe Cisco native interfaces
7. gNMI Subscribe OpenConfig interfaces

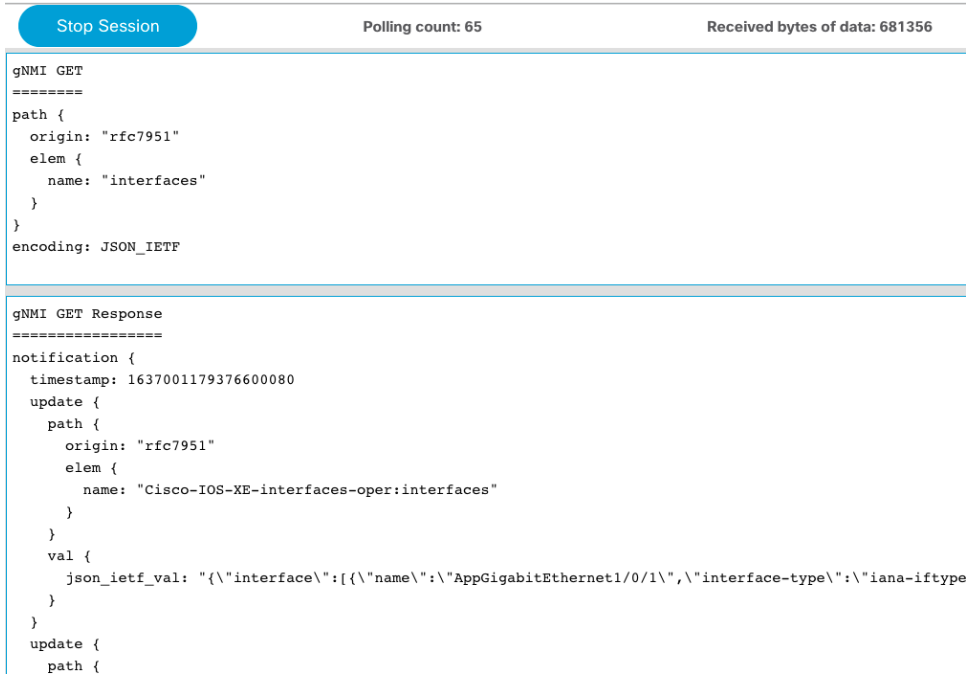
Certificate Authentication within YANG Suite:



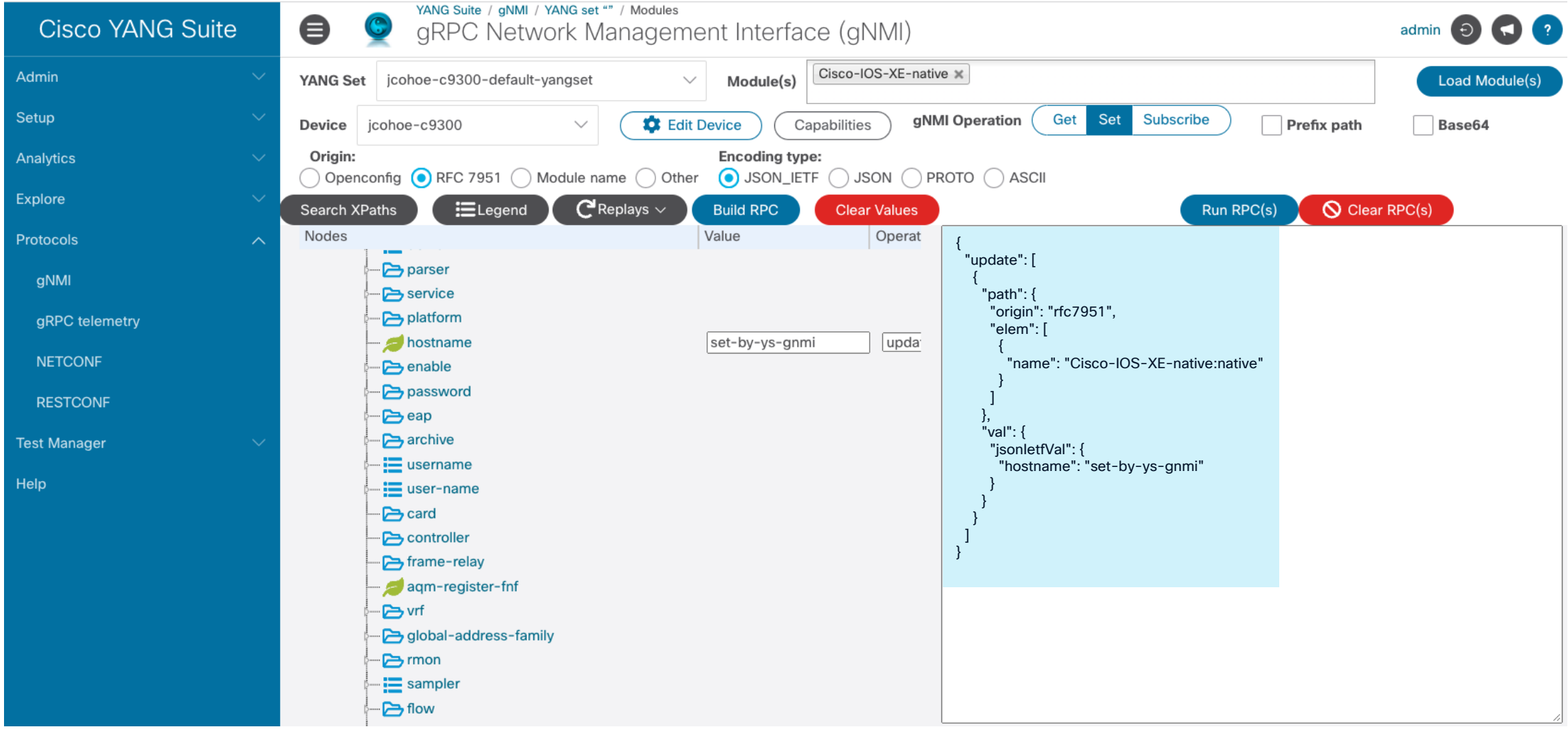


gNMI GET for Cisco Native hostname

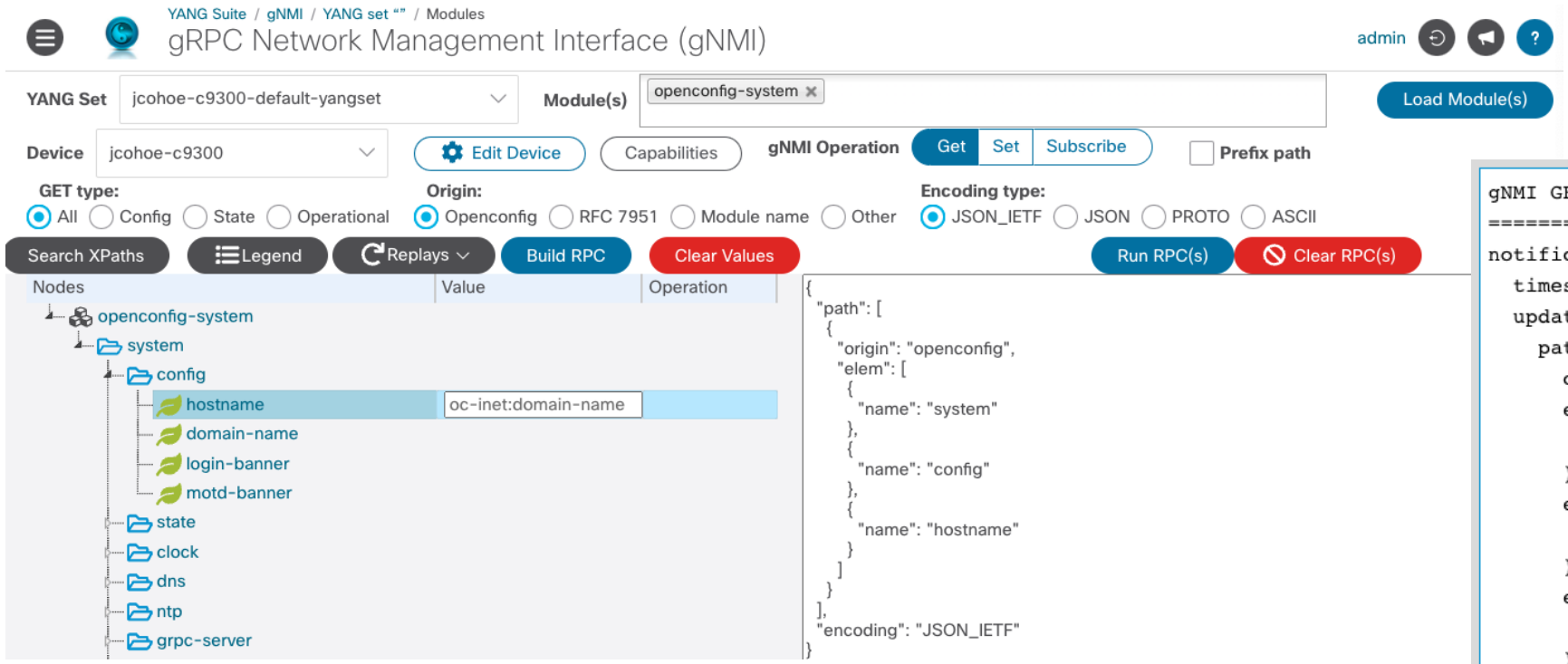




gNMI SET for Cisco Native hostname



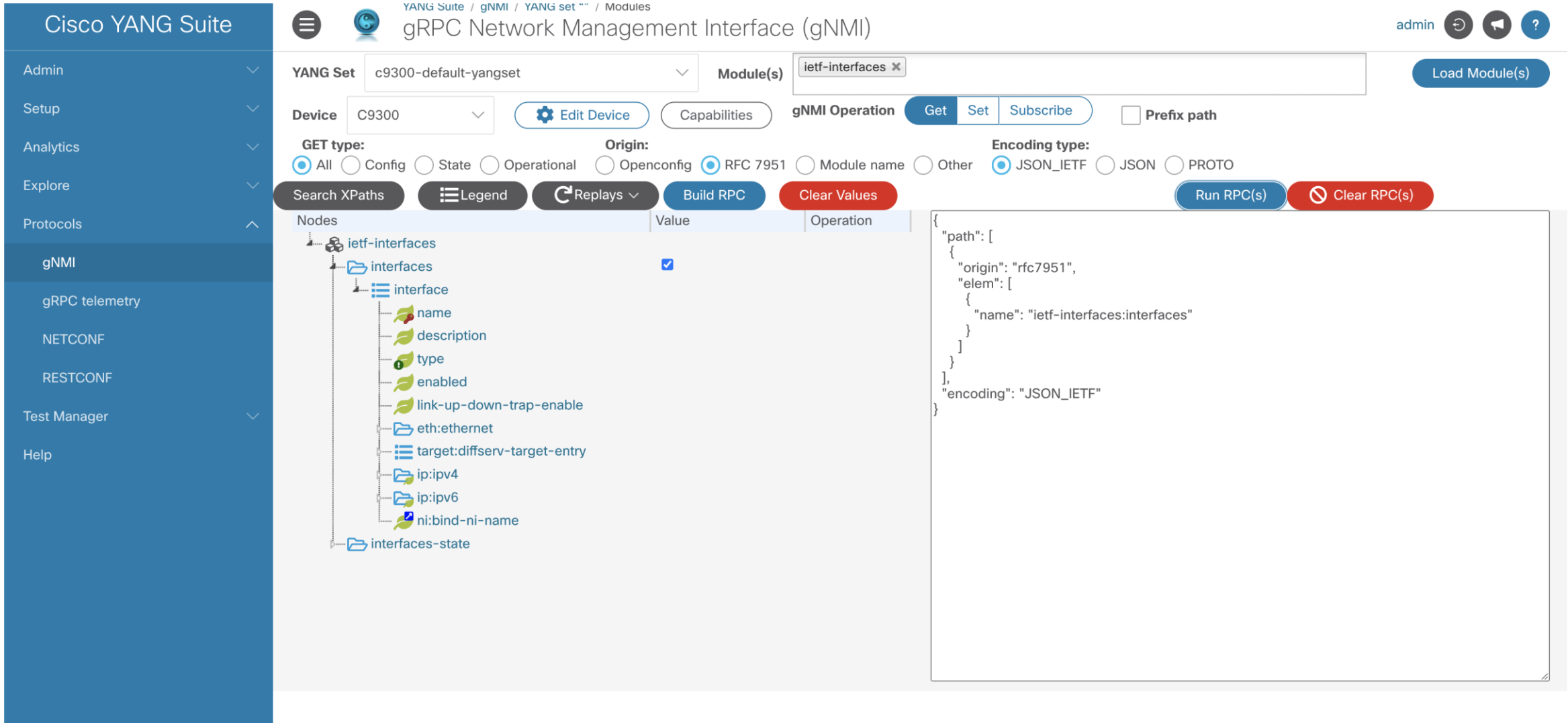
gNMI GET OC Hostname




```
gNMI GET Response
=================
notification {
  timestamp: 1663716002850520766
  update {
    path {
      origin: "openconfig"
      elem {
        name: "system"
      }
      elem {
        name: "config"
      }
      elem {
        name: "hostname"
      }
    }
    val {
      json_ietf_val: "\"set-by-ys-gnmi-ansible\""
    }
  }
}
```

```
JSON Decoded
============
"set-by-ys-gnmi-ansible"
```

gNMI GET IETF Interfaces



gNMI GET IETF Interfaces Response

Stop Session | Session status: running | Received bytes of data: 14432

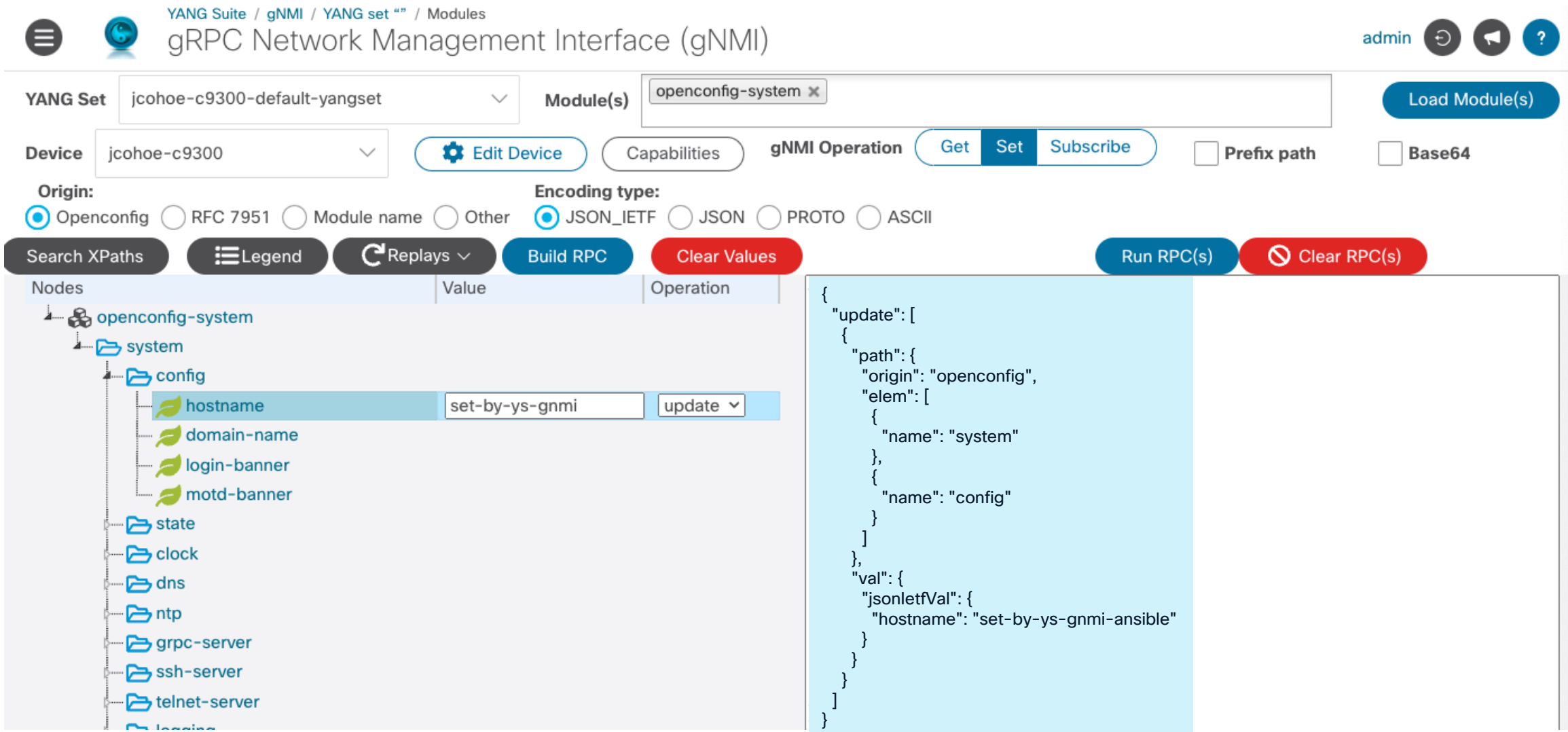
```
Connecting insecure channel
```

```
gNMI GET
========
path {
  origin: "rfc7951"
  elem {
    name: "ietf-interfaces:interfaces"
  }
}
encoding: JSON_IETF
```

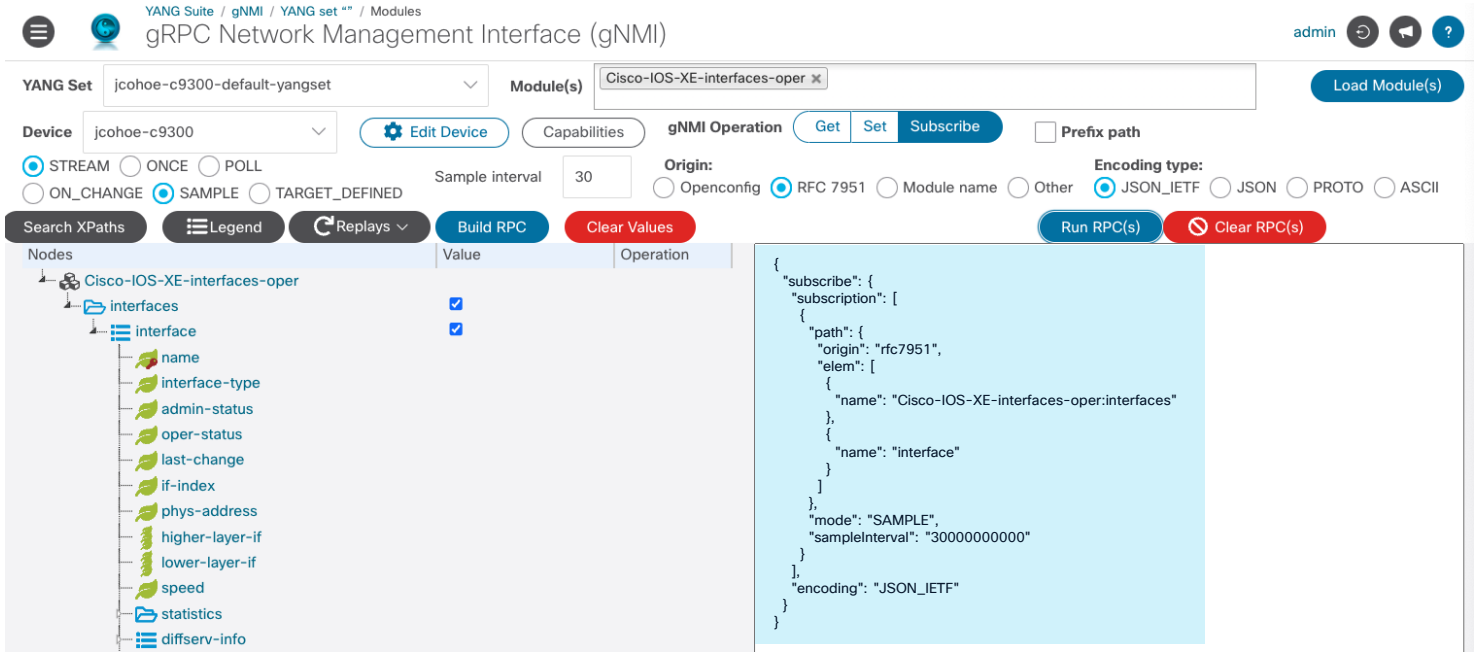
```
gNMI GET Response
=================
notification {
  timestamp: 1661543913969767107
  update {
    path {
      origin: "rfc7951"
      elem {
        name: "ietf-interfaces:interfaces"
      }
    }
    val {
      json_ietf_val: "{\"interface\":[{\"name\":\"FortyGigabitEthernet1/1/1\",\"type\":\"iana-if-type:ethernetCsmacd\",\"enabled\":true,\"ietf-ip:ipv4\":{},\"ietf-ip
    }
  }
}
```

```
JSON Decoded
============
{
  "interface": [
    {
      "name": "FortyGigabitEthernet1/1/1",
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "ietf-ip:ipv4": {},
      "ietf-ip:ipv6": {}
    },
    {
      "name": "FortyGigabitEthernet1/1/2",
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "ietf-ip:ipv4": {},
      "ietf-ip:ipv6": {}
    },
```

gNMI SET OC Hostname



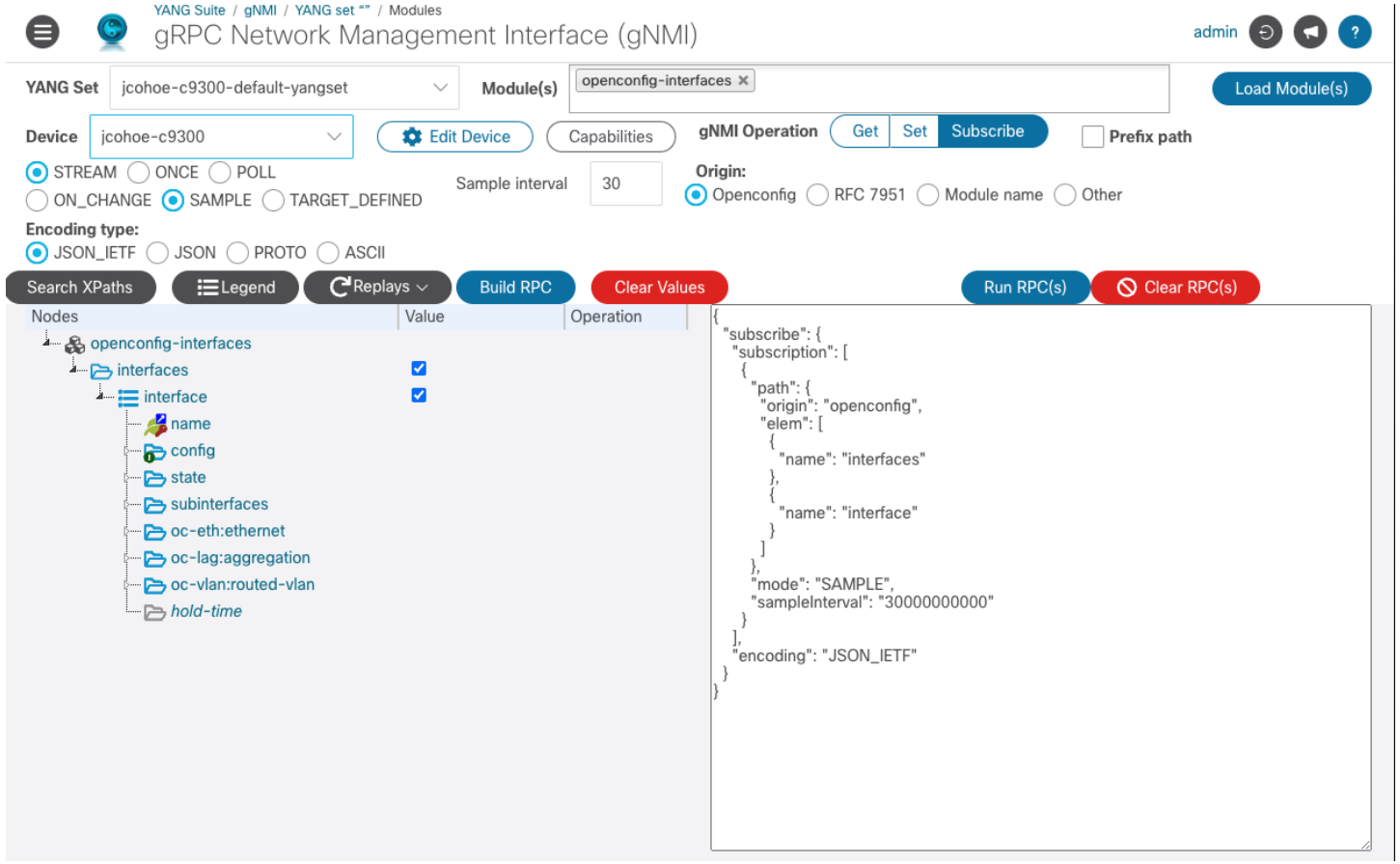
gNMI Subscribe Cisco native interfaces




```
update {
  timestamp: 1663716246800647000
  update {
    path {
      origin: "rfc7951"
      elem {
        name: "Cisco-IOS-XE-interfaces-oper:interfaces"
      }
      elem {
        name: "interface"
      }
    }
    val {
      json_ietf_val: "[{\"name\":\"AppGigabitEthernet1/0/1\"
    }
  }
}
```

```
JSON Decoded
============
[
  {
    "name": "AppGigabitEthernet1/0/1",
    "interface-type": "iana-iftype-ethernet-csmacd",
    "admin-status": "if-state-up",
    "oper-status": "if-oper-state-ready",
    "last-change": "2022-07-26T17:43:57.920000+00:00",
    "if-index": 49,
    "phys-address": "70:1f:53:9b:0f:a9",
    "speed": "1000000000",
    "statistics": {
      "discontinuity-time": "2022-07-26T17:40:57+00:00",
      "in-octets": "0",
      "in-unicast-pkts": "0",
      "in-broadcast-pkts": "0",
      "in-multicast-pkts": "0",
      "in-discards": 0,
      "in-errors": 0,
      "in-unknown-protos": 0,
      "out-octets": 4073505065,
      "out-unicast-pkts": "35917940",
      "out-broadcast-pkts": "17172777",
      "out-multicast-pkts": "17949523",
      "out-discards": "0",
      "out-errors": "0",
      "rx-pps": "0",
      "rx-kbps": "0",
      "tx-pps": "5",
      "tx-kbps": "4",
      "num-flaps": "0",
      "in-crc-errors": "0",
      "in-discards-64": "0",
      "in-errors-64": "0",
      "in-unknown-protos-64": "0",
      "out-octets-64": "4073505065"
    },
```

gNMI Subscribe OpenConfig interfaces




```
update {
  timestamp: 1663716380431805000
  update {
    path {
      origin: "openconfig"
      elem {
        name: "interfaces"
      }
      elem {
        name: "interface"
      }
    }
    val {
      json_ietf_val
    }
  }
}
```

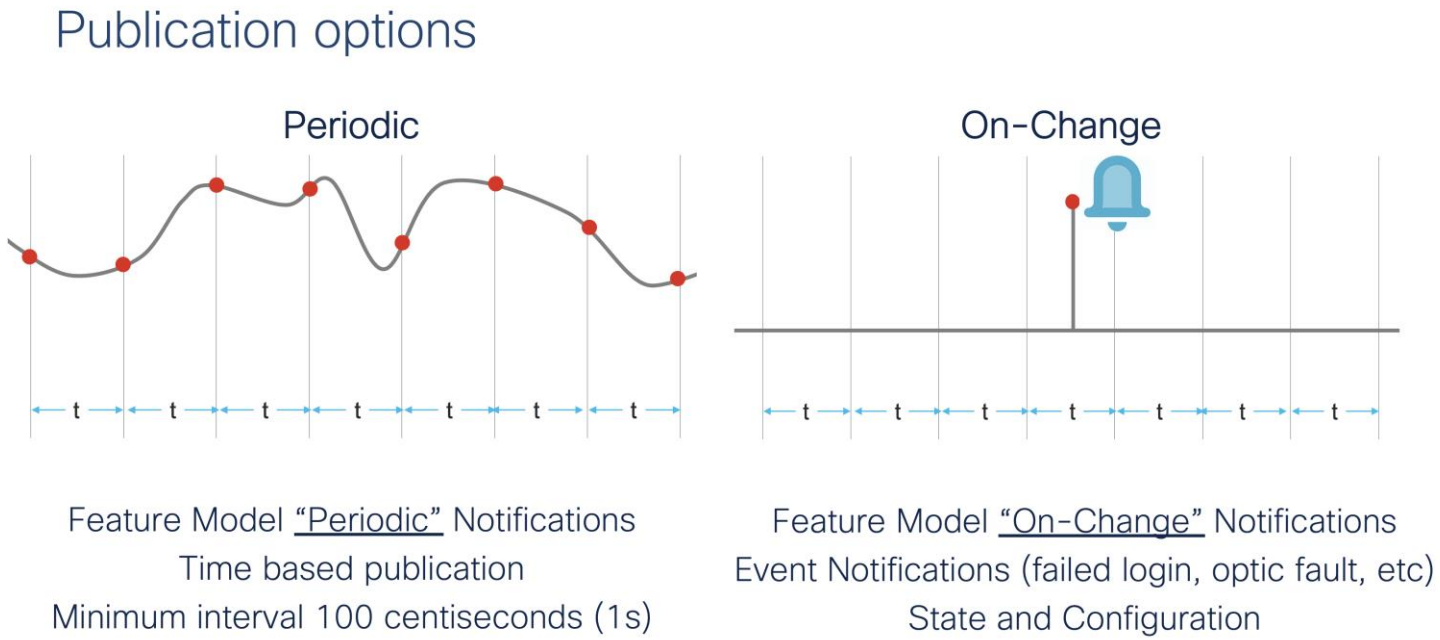
```
JSON Decoded
============
[
  {
    "name": "FortyGigabitEthernet1/1/1",
    "config": {
      "name": "FortyGigabitEthernet1/1/1",
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true
    },
    "state": {
      "name": "FortyGigabitEthernet1/1/1",
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "ifindex": 45,
      "admin-status": "UP",
      "oper-status": "NOT_PRESENT",
      "last-change": "1658857381366000000",
      "counters": {
        "in-octets": "0",
        "in-unicast-pkts": "0",
        "in-broadcast-pkts": "0",
        "in-multicast-pkts": "0",
        "in-discards": "0",
        "in-errors": "0",
        "in-unknown-protos": "0",
        "in-fcs-errors": "0",
        "out-octets": "0",
        "out-unicast-pkts": "0",
        "out-broadcast-pkts": "0",
        "out-multicast-pkts": "0",
        "out-discards": "0",
        "out-errors": "0",
        "last-clear": "1658857257000000000"
      },
      "openconfig-platform-port:hardware-port": "FortyGigabitEthernet1/1/1"
    },
```

Telemetry

gNMI on-change

Periodic vs. on-change

Previously, we supported yang-notification on NETCONF, gRPC and a selection of gNMI for on-change telemetry subscriptions. This allows us to receive a notification each time there is a modification to the current device config

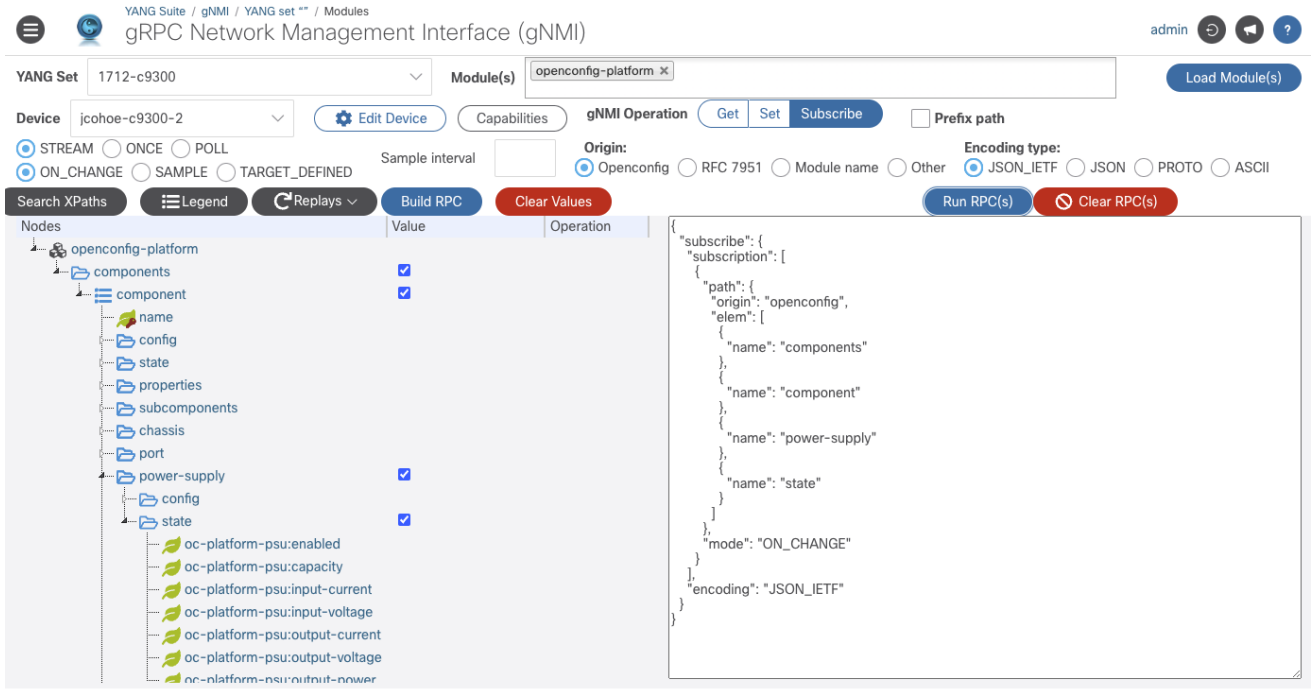


gNMI on-change

17.6+
17.10+

The Update Trigger of “On-Change” is supported for gNMI with specific xpaths

Query the MDT-Capabilities-Oper.yang to understand which xpaths are supported




```
Processing returns...
```

```
JSON Decoded
============
{
  "openconfig-platform-psu:enabled": "false"
}
```

```
Xpath/Value
===========
[{'datatype': 'key',
  'value': 'PowerSupply1/B',
  'xpath': '/openconfig-platform:components/component/name'}]
```

```
JCOHOE-C9300-2#sh tel ie sub 2147483649 detail
Telemetry subscription detail:

  Subscription ID: 2147483649
  Type: Dynamic
  State: Valid
  Stream: yang-push
  Filter:
    Filter type: xpath
    XPath: /oc-platform:components/component/power-supply/state
  Update policy:
    Update Trigger: on-change
    Synch on start: Yes
    Dampening period: 0
  Encoding: encode-kvgpb
  Source VRF:
  Source Address:
  Notes: Subscription validated

  Legacy Receivers:
    Address                                       Port     Protocol         Protocol Profile
    ------------------------------------------------------------------------------------------
    10.85.134.103                                 51128    gNMI
```

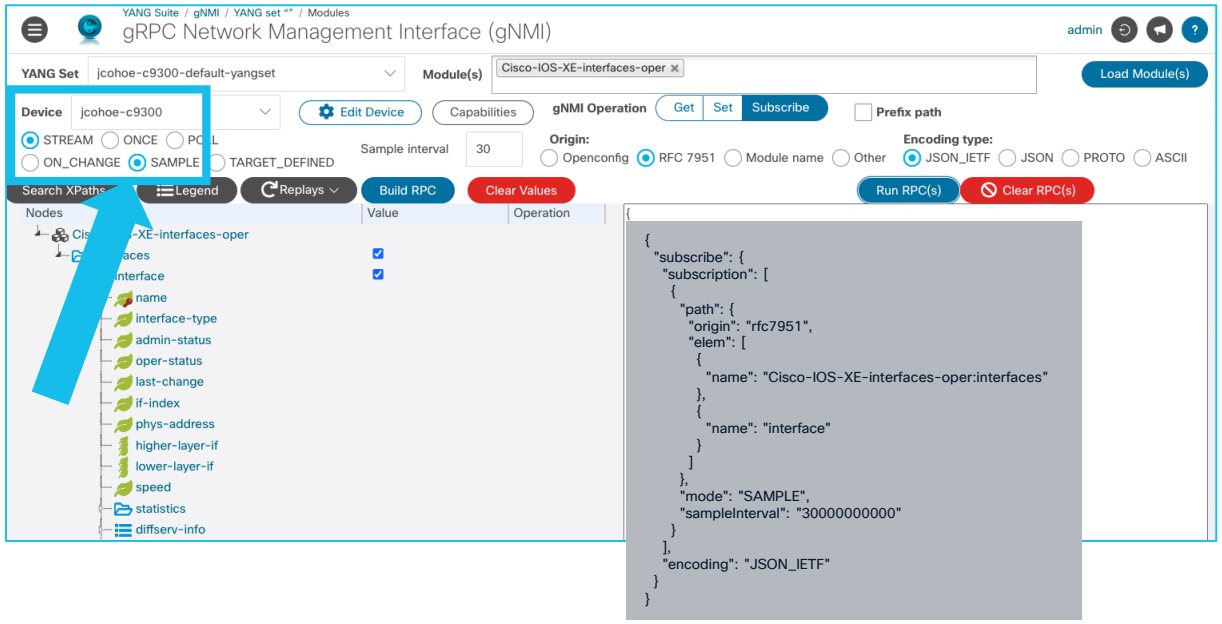
https://github.com/jeremycohoe/cisco-ios-xe-mdt/blob/master/mdt-capabilities-c9300-1710.xml
https://github.com/YangModels/yang/blob/main/vendor/cisco/xe/17101/Cisco-IOS-XE-mdt-capabilities-oper.yang
Guide @ https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1710/b_1710_programmability_cg/m_1710_prog_ietf_telemetry.html#id_114707

gNMI - STREAM subscriptions with on_change

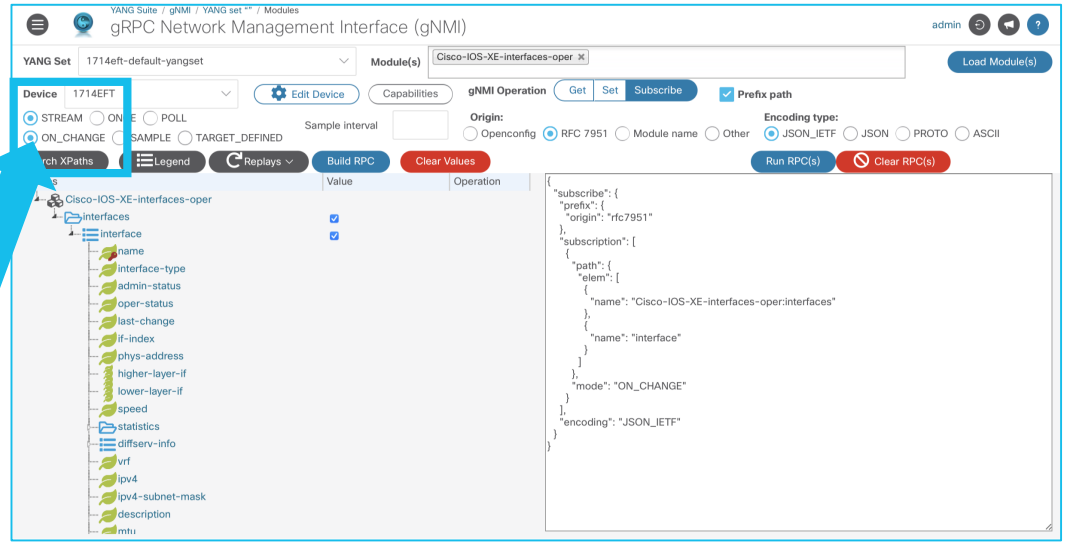
17.14

- Previously the gNMI SUBSCRIBE operation supported only a very few on-change models
- Now gNMI on-change subscribe supports the same Xpaths as NETCONF and gRPC.

gNMI Steam Sample (periodic) <17.14



gNMI Stream on-change 17.14

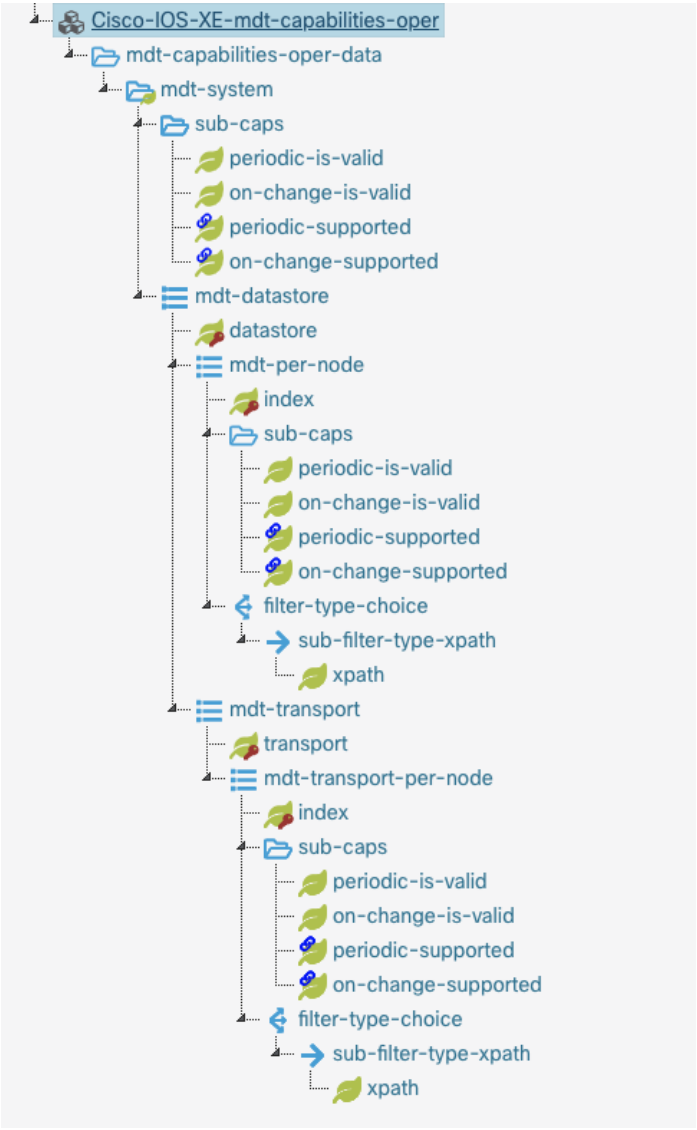


Cisco-IOS-XE-mdt-capabilities-oper

Details about both on-change and periodic subscription support is detailed in this data model

Module	Cisco-IOS-XE-mdt-capabilities-oper
Revision	2022-11-01
Revision Info	Update yang-version to 1.1
Description	This module contains a collection of YANG definitions for advertising the notification capabilities of the system with respect to streaming telemetry on the YANG-Push stream. Capabilities can be specified at the system level, or for specific nodes (and their children) of a specific datastore. Capabilities specified on a per-node basis override the system level capabilities. Capabilities can also vary by the transport that is used to subscribe for notifications. A node may support a particular capability in general, but also have transport-specific exceptions. If the user wishes to find the value of a capability for a particular node, they should: 1) Search for the desired datastore in the mdt-datastore-capabilities list. 2) If an entry is found, search for the desired transport in the mdt-transport-capabilities list (if the transport-specific capabilities are desired). If the transport is found, iterate through the per-node-capabilities entries for the transport, in the order they appear in the list. The first entry that has an "is-valid" leaf for the capability, and has a filter selecting the desired node, specifies the capability value. 3) If the capability is not found for the transport (or if the transport-independent capabilities are desired), iterate through the per-node-capabilities list for the datastore, in the same manner as in step 2. 4) If the capability value is not found, use the system-level value for the capability, if it has an "is-valid" leaf. 5) If the capability is not found in the previous steps, then the system does not specify a value for that capability. Copyright (c) 2021-2022 by Cisco Systems, Inc. All rights reserved.

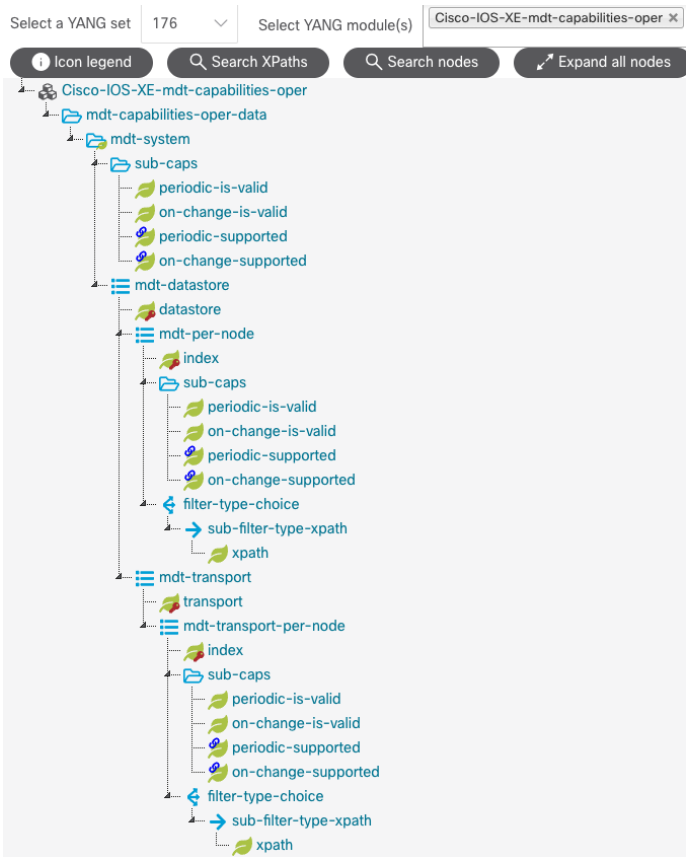

```
JSON Decoded
============
{
  "mdt-system": {
    "sub-caps": {
      "periodic-is-valid": [
        null
      ],
      "on-change-is-valid": [
        null
      ],
      "periodic-supported": "mdt-cap-notif-config mdt-cap-notif-state",
      "on-change-supported": [
        null
      ]
    },
    "mdt-datastore": [
      {
        "datastore": "mdt-cap-ds-running",
        "mdt-per-node": [
          {
            "index": 3,
            "sub-caps": {
              "on-change-is-valid": [
                null
              ],
              "on-change-supported": "mdt-cap-notif-config mdt-cap-notif-state"
            },
            "xpath": "/xcvr-ios-xe-oper:transceiver-oper-data"
          },
          {
            "index": 54,
            "sub-caps": {
              "on-change-is-valid": [
                null
              ],
              "on-change-supported": "mdt-cap-notif-config mdt-cap-notif-state"
```

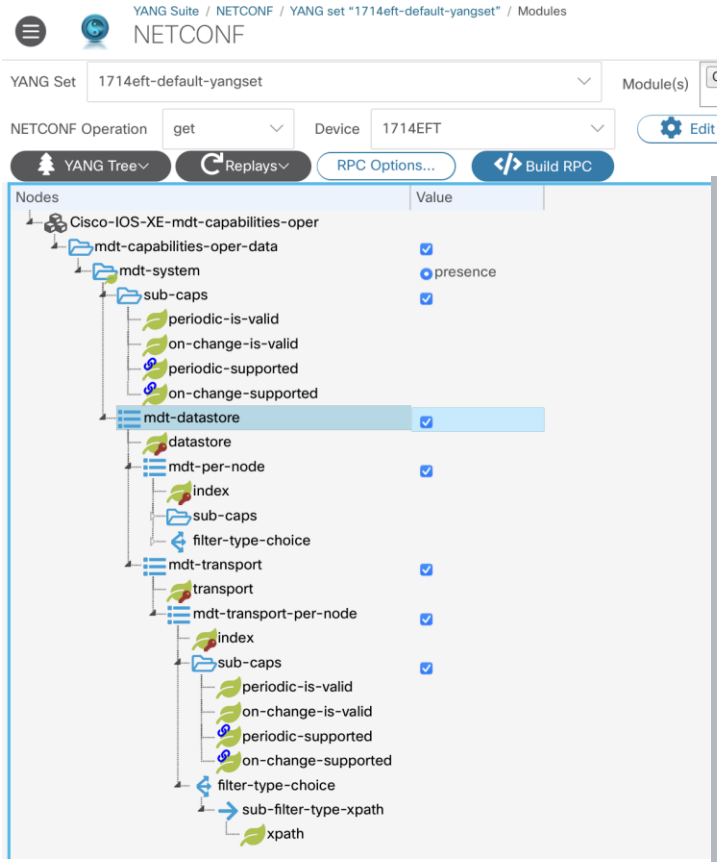



https://github.com/YangModels/yang/blob/main/vendor/cisco/xe/17111/Cisco-IOS-XE-mdt-capabilities-oper.yang

Cisco-IOS-XE-mdt-capabilities-oper

Example of reading the model to see data about support for both on-change and periodic subscriptions





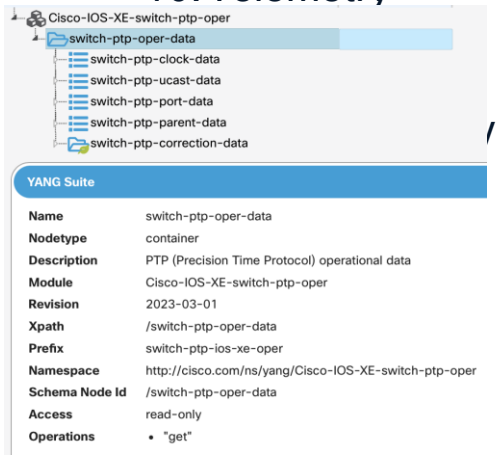

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <get>
    <filter>
      <mdt-capabilities-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-capabilities-oper">
        <mdt-system>
          <sub-caps/>
          <mdt-datastore>
            <mdt-per-node/>
            <mdt-transport>
              <mdt-transport-per-node>
                <sub-caps/>
              </mdt-transport-per-node>
              <mdt-transport-per-node/>
            </mdt-transport>
          </mdt-datastore>
        </mdt-system>
      </mdt-capabilities-oper-data>
    </filter>
  </get>
</rpc>
```

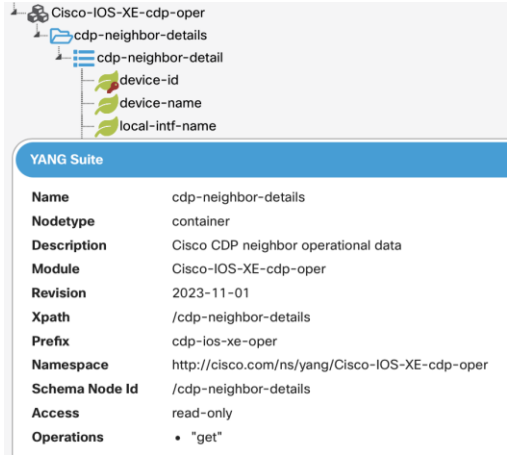
Xpaths supported with mdt-capabilities-oper

17.14 Switching

NETCONF, gRPC and gNMI now have Xpath feature parity for on-change subscriptions

1. Transceiver
2. VLAN
3. Trustsec
4. TCAM
5. PTP
6. Stacking
7. POE
8. Platform components
9. PSU State
10. Telemetry
14. DNS
15. CDP
16. Boot
17. Breakout Port
18. AppHosting





No on-change support for OpenConfig models over NETCONF

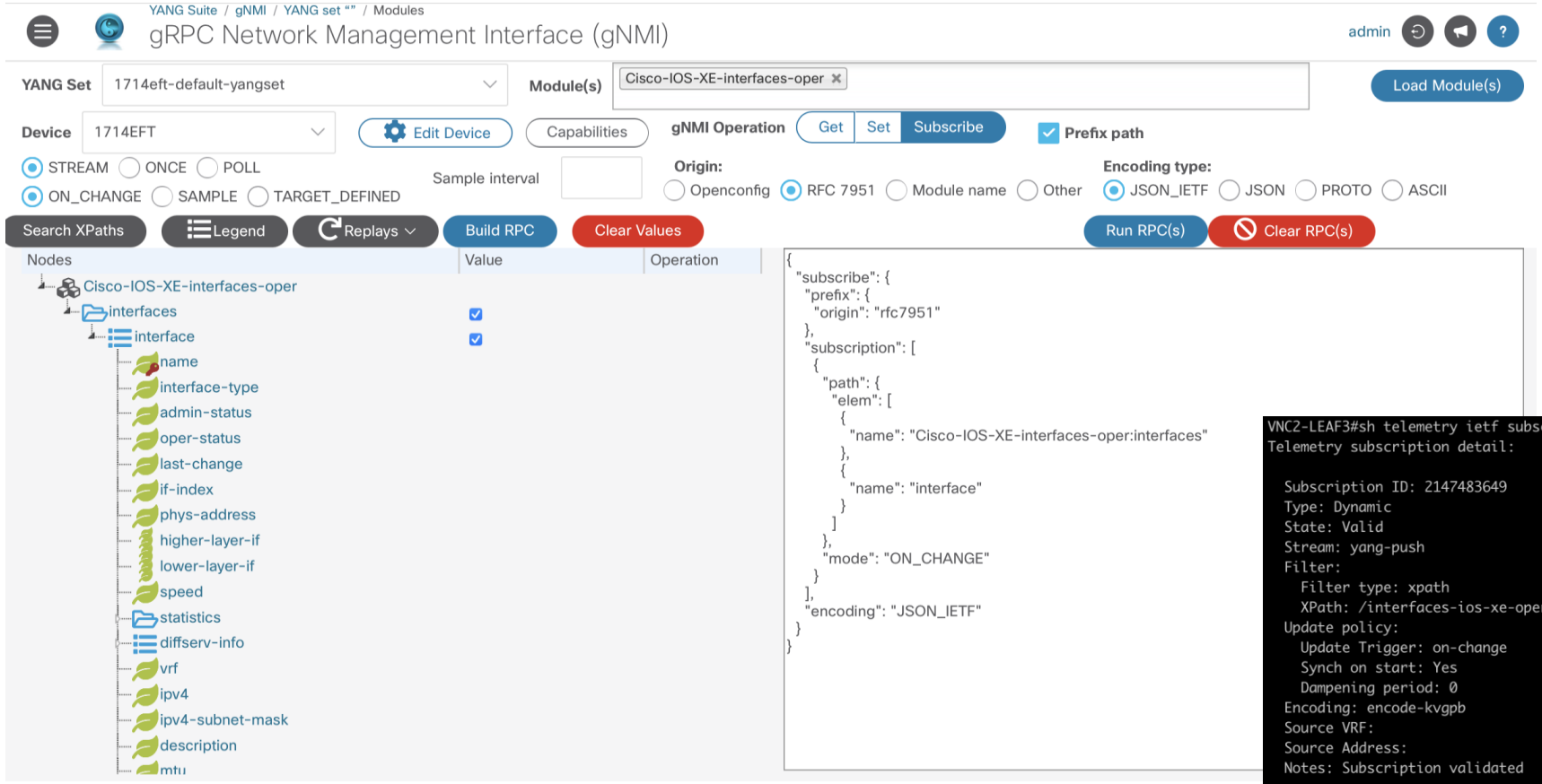
1. /app-hosting-ios-xe-cfg:app-hosting-cfg-data
2. /app-hosting-ios-xe-oper:app-hosting-oper-data
3. /bc-port-ios-xe-oper:breakout-port-oper-data
4. /boot-integrity-ios-xe-oper:boot-integrity-oper-data
5. /cdp-ios-xe-oper:cdp-neighbor-details
6. /dns-ios-xe-oper:dns-oper-data
7. /ha-ios-xe-oper:ha-oper-data
8. /interfaces-ios-xe-oper:interfaces/interface
9. /interfaces-ios-xe-oper:interfaces/interface/diffserv-info
10. /interfaces-ios-xe-oper:interfaces/interface/lag-aggregate-state
11. /linecard-ios-xe-oper:linecard-oper-data
12. /mdt-cfg:mdt-config-data
13. /mdt-oper-v2:mdt-oper-v2-data
14. /meraki-connect-ios-xe-oper:meraki-connect-oper-data
15. /oc-platform:components/component/power-supply/state
16. /oc-platform:components/component/power-supply/state

17.10+

17. /platform-ios-xe-oper:components
18. /poe-ios-xe-oper:poe-oper-data
19. /stacking-ios-xe-oper:stacking-oper-data
20. /switch-ptp-ios-xe-oper:switch-ptp-oper-data
21. /tcam-ios-xe-oper:tcam-details - Deprecated
22. /trustsec-ios-xe-oper:trustsec-state/cts-rolebased-sgtmaps
23. /trustsec-ios-xe-oper:trustsec-state/cts-sxp-connections
24. /vlan-ios-xe-oper:vlans
25. /xcvr-ios-xe-oper:transceiver-oper-data

gNMI on-change example – interfaces-oper

17.14




```
VNC2-LEAF3#sh telemetry ietf subscription 2147483649 detail
Telemetry subscription detail:

  Subscription ID: 2147483649
  Type: Dynamic
  State: Valid
  Stream: yang-push
  Filter:
    Filter type: xpath
    XPath: /interfaces-ios-xe-oper:interfaces/interface
  Update policy:
    Update Trigger: on-change
    Synch on start: Yes
    Dampening period: 0
  Encoding: encode-kvgpb
  Source VRF:
  Source Address:
  Notes: Subscription validated

  Named Receivers:
    Name                                          Last State Change  State                Explanation
    ----------------------------------------------------------------------------------------------------
    gNMI://10.85.134.103:51740                    03/14/24 14:40:27  Connected
```

gNOI

Operational API

gNOI – gRPC Network Operations Interface

1. gRPC Network Operations Interface, or gNOI, is a set of gRPC-based microservices, used for executing operational commands on network devices
2. gNOI operations are executed against the gNMI API interface
3. gNOI is defined and implemented on a per proto basis
4. There are many protos defined - some are more mature and evolve and different pace

Protobuf RPC	Use	Related CLI	Release
Cert.proto	TLS Certificate management	crypto pki ...	17.3
Os.proto	Network Operating System management	install add file ...	17.5
Reset.proto	Factory Reset and secure wipe	write erase ...	17.7
File.proto	Not implemented on IOS XE, use YANG	copy, delete	N/A
System.proto	Not implemented on IOS XE, use YANG	reload, set boot	N/A

https://github.com/openconfig/gnoi
https://github.com/jeremycohoe/cisco-catalyst-gnoi-reset.proto Factory Reset API Lab Guide
https://github.com/jeremycohoe/cisco-catalyst-gnoi-cert.proto Certificate Management API Lab Guide
https://www.youtube.com/watch?v=cGxgBAlefOQ GNOI OS.proto - Operating System API demo

gNOI

OS.proto

gNOI os.proto – Operating System API

OS installation, activation, and verification API
https://github.com/google/gnxi/tree/master/gnoi_os

gNOI defines a set of gRPC-based microservices for executing operational commands on network devices. OS **Install**, **Activate**, and **Verification** are defined and addressed here:
https://github.com/openconfig/gnoi/blob/master/os/os.proto

The OS service provides an interface for OS installation on a Target.
The Client progresses through 3 RPCs:
1) Installation – provide the Target with the OS package.
2) Activation – activate an installed OS package.
3) Verification – verify the installed and activated version

Additional CLI: show gnxi **os**

```
C9300#show gnxi os ?
  activate  OS activate operations
  detail    Detailed overview of OS operations
  install   OS install operations
  internal  Internal OS commands
  summary   Summary of OS operations
```

gNOI OS Client

A simple shell binary that performs OS client operations against a gNOI target.

gNOI OS Client Operations

- `-op install` installs the provided OS image onto the target.
- `-op activate` tells the target to boot into the specified OS version on next reboot.
- `-op verify` verifies the version of the OS currently running on the target.

Install

```
go get github.com/google/gnxi/gnoi_os
go install github.com/google/gnxi/gnoi_os
```

Run

```
./gnoi_os \
    -target_addr localhost:9339 \
    -target_name target.com \
    -ca ca.crt \
    -key client.key \
    -cert client.crt \
    -version 1.1 \
    -os myosfile.img \
    -op install | activate | verify
```

Bundle mode will be converted to Install at reboot

GNOI OS.proto demo

```
jcohoe@jcohoe-ubuntu18-lab:~$ cd ~/certs-jcohoe-c9300-2/ ; gnoi_os -insecure -target_addr 10.85.134.92:9339 -op install -target_name c9300 -alsologtostderr -cert ./cli
ent.crt -ca ./rootCA.pem   -key ./rootCA.key -version 17.06.01.0.135639.1618187331 -time_out 999s -os /tftpboot/cat9k_iosxe.17.06.01-20210411.bin
jcohoe@jcohoe-ubuntu18-lab:~/certs-jcohoe-c9300-2$ cd
jcohoe@jcohoe-ubuntu18-lab:~$
jcohoe@jcohoe-ubuntu18-lab:~$ cd ~/certs-jcohoe-c9300-2/ ; gnoi_os -insecure -target_addr 10.85.134.92:9339 -op activate -target_name c9300 -alsologtostderr -cert ./cl
ient.crt -ca ./rootCA.pem   -key ./rootCA.key -version 17.06.01.0.135639.1618187331 -time_out 999s -os /tftpboot/cat9k_iosxe.17.06.01-20210411.bin
jcohoe@jcohoe-ubuntu18-lab:~/certs-jcohoe-c9300-2$
jcohoe@jcohoe-ubuntu18-lab:~/certs-jcohoe-c9300-2$ cd ~/certs-jcohoe-c9300-2/ ; gnoi_os -insecure -target_addr 10.85.134.92:9339 -op verify -target_name c9300 -alsolog
tostderr -cert ./client.crt -ca ./rootCA.pem   -key ./rootCA.key
I0418 11:57:10.555504   29388 gnoi_os.go:98] Running OS version: 17.06.01.0.135639.1618187331
jcohoe@jcohoe-ubuntu18-lab:~/certs-jcohoe-c9300-2$
```

```
telnet
C9300#
*Apr 18 18:29:46.613: %INSTALL-5-INSTALL_START_INFO: Switch 1 R0/0: install_engine: Started install remove
*Apr 18 18:32:09.394: %PLATFORM_FEP-1-FRU_PS_ACCESS: Switch 1: power supply A is not responding
*Apr 18 18:32:11.246: %INSTALL-5-INSTALL_COMPLETED_INFO: Switch 1 R0/0: install_engine: Completed install remove
*Apr 18 18:35:24.621: %INSTALL-5-INSTALL_START_INFO: Switch 1 R0/0: install_engine: Started install add flash:gNOI_iosxe_17.06.01.0.135639.1618187331.bin
*Apr 18 [redacted]: %INSTALL-5-INSTALL_COMPLETED_INFO: Switch 1 R0/0: install_engine: Completed install add PACKAGE flash:gNOI_iosxe_17.06.01.0.135639.1618187331.bin
*Apr 18 18:38:58.711: %ISSU-3-ISSU_COMP_CHECK_FAILED: Switch 1 R0/0: install_engine: ISSU compatibility check failed for 17.06.01.0.135639
C9300#
C9300#
*Apr 18 [redacted]: %INSTALL-5-INSTALL_START_INFO: Switch 1 R0/0: install_engine: Started install activate
*Apr 18 18:44:28.608: %INSTALL-5-INSTALL_AUTO_ABORT_TIMER_PROGRESS: Switch 1 R0/0: install_mgr: Install auto abort timer will expire in 7200 seconds
```

```
C9300#show gnxi os summary
Mode: Install Mode
Current Operation: No operation in progress
No image has been added as Inactive
Current Activated Version: 17.05.01.0.144.1617180620
```

```
C9300#show gnxi os summary
Mode: Install Mode
Current Operation: No operation in progress
No image has been added as Inactive
Current Activated Version: 17.06.01.0.135639.1618187331
Last Executed RPC: Verify, Success
```

Demo video on YouTube

Verify:
cd ~/certs-jcohoe-c9300-2/ ; gnoi_os -insecure -target_addr 10.85.134.92:9339 -op verify -target_name c9300 -alsologtostderr -cert ./client.crt -ca ./rootCA.pem -key ./rootCA.key
Running OS version: 17.05.01.0.144.1617180620

Install:
cd ~/certs-jcohoe-c9300-2/ ; gnoi_os -insecure -target_addr 10.85.134.92:9339 -op install -target_name c9300 -alsologtostderr -cert ./client.crt -ca ./rootCA.pem -key ./rootCA.key -version 17.06.01.0.135639.1618187331 -time_out 999s -os /tftpboot/cat9k_iosxe.17.06.01-20210411.bin

Activate:
cd ~/certs-jcohoe-c9300-2/ ; gnoi_os -insecure -target_addr 10.85.134.92:9339 -op activate -target_name c9300 -alsologtostderr -cert ./client.crt -ca ./rootCA.pem -key ./rootCA.key -version 17.06.01.0.135639.1618187331 -time_out 999s -os /tftpboot/cat9k_iosxe.17.06.01-20210411.bin

Verify:
Running OS version: 17.06.01.0.135639.1618187331

show CLI: show gnxi os summary

gNOI

cert.proto

gNOI cert.proto – “Certificate Management API”

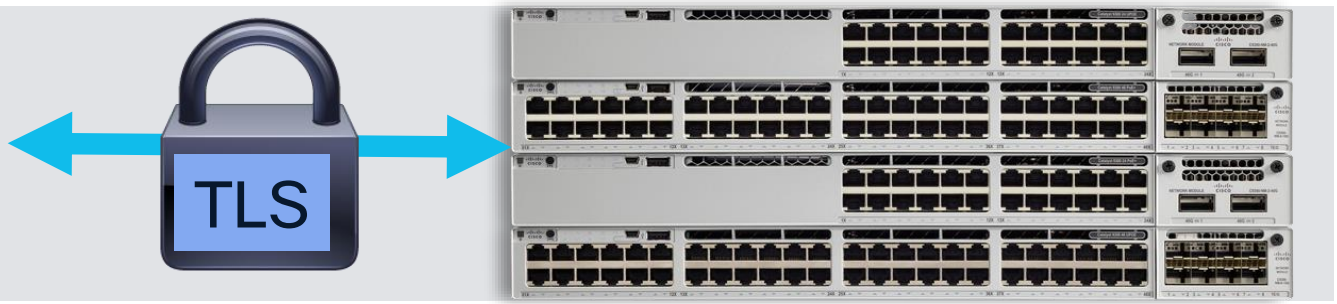
Described at https://github.com/openconfig/gnoi/blob/master/cert/cert.proto
“This file defines the gNOI API to be used for certificate installation and rotation”

gRPC Network Operations Interface (gNOI) defines a set of gRPC-based microservices for executing operational commands on network devices.

The Certificate Management Service primarily exports two main RPCs, Install and Rotate, that are used for the installation of new certificates, and rotation of existing certificates on a device

Supported operations are provision, install, rotate, revoke, get, and check

Tooling is used to install the certificates onto the network device
This can be used instead of “crypto pki” CLI or “Cisco-IOS-XE-crypto-RPC.YANG”



Install certificate for gNMI

gnoi_cert -insecure -target_addr 10.1.1.5:9339 -op provision -target_name c9300 -alsologtostderr -organization "jcohoe org" -ip_address 10.1.1.5 -time_out=10s -min_key_size=2048 -cert_id gnxi-cert -state BC -country CA -ca ./rootCA.pem -ca_key ./rootCA.key

```
auto@automation:~/gnmi_ssl/certs$ gnoi_cert -insecure -target_addr 10.1.1.5:9339 -op provision -ca ./rootCA.pem -key ./rootCA.key -target_name c9300 -al
sologtostderr -organization "jcohoe org" -ip_address 10.1.1.5 -time_out=10s -min_key_size=2048 -cert_id gnxi-cert -state BC -country CA -ca_key ./rootCA
.key

I0917 15:25:28.969060   17154 gnoi_cert.go:161] Install success
auto@automation:~/gnmi_ssl/certs$
```

The gNMI service is restarted as the new certificate is applied

```
Sep 18 02:35:15.317: %PKI-6-TRUSTPOINT_CREATE: Trustpoint: gnxi-cert created succesfully
Sep 18 02:35:15.317: %PKI-4-NOAUTOSAVE: Configuration was modified.  Issue "write memory" to save new certificate
Sep 18 02:35:16.750: %CRYPTO_ENGINE-5-KEY_ADDITION: A key named gnxi-cert has been generated or imported by crypto-engine
Sep 18 02:35:16.795: %PKI-4-NOCONFIGAUTOSAVE: Configuration was modified.  Issue "write memory" to save new IOS PKI configuration
Sep 18 02:35:17.382: %PKI-4-NOCONFIGAUTOSAVE: Configuration was modified.  Issue "write memory" to save new IOS PKI configuration
Sep 18 02:35:17.395: %SYS-5-CONFIG_P: Configured programmatically by process iosp_vty_100001_gnmib_fd_238 from console as NETCONF on vty4294967295
Sep 18 02:35:17.399: %SYS-5-CONFIG_P: Configured programmatically by process iosp_vty_100001_gnmib_fd_238 from console as NETCONF on vty4294967295
Sep 18 02:35:17.403: %SYS-5-CONFIG_P: Configured programmatically by process iosp_vty_100001_gnmib_fd_238 from console as NETCONF on vty4294967295
Sep 18 02:35:17.553: %PKI-6-TRUSTPOOL_DOWNLOAD_SUCCESS: Trustpool Download is successful
Sep 18 02:35:17.553: %PKI-4-NOAUTOSAVE: Configuration was modified.  Issue "write memory" to save new certificate
Sep 18 02:35:17.403: %GNMIB-5-SRV_OPER_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] operational status: DOWN
Sep 18 02:35:18.356: %GNMIB-5-SRV_ADMIN_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] administrative state: DISABLED
Sep 18 02:35:18.356: %GNMIB-5-SRV_ADMIN_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] administrative state: ENABLED
Sep 18 02:35:23.360: %GNMIB-5-SRV_OPER_SCN: Switch 1 R0/0: gnmib: Component [ gnmi::broker ] operational status: UP
```

gNOI cert.proto demo

1. Device has no previous gNXI configuration or certificates
2. Enable gNMI and secure-init to create certificates
3. Send the gNMI certificate to the device
4. Send another certificate to the device for secure telemetry
5. Check with the GET operation that both certificates are installed

auto@automation: ~ (ssh)

```
C9300#show run | i gnmi
C9300#show run | i gnxi
C9300#show crypto pki trustpoints | i Tr
Trustpoint SLA-TrustPoint:
Trustpoint TP-self-signed-2903776758:
Trustpoint CISCO_IDEVID_SUDI_LEGACY:
Trustpoint CISCO_IDEVID_SUDI_LEGACY0:
Trustpoint CISCO_IDEVID_SUDI:
Trustpoint CISCO_IDEVID_SUDI0:
C9300#
C9300#
```

auto@automation: ~/gnmi_ssl (ssh)

```
auto@automation:~/gnmi_ssl$
```

gNOI

reset.proto

gNOI reset.proto

17.7

Factory reset using a single API call is used as part of RMA, re-provisioning, security, and ZTP workflows for automation
This gNOI API is described at https://github.com/openconfig/gnoi/blob/master/factory_reset/
with tooling from https://github.com/google/gnxi/tree/master/gnoi_reset

- gNOI defines a set of gRPC-based microservices for executing operational commands on network devices. Reset has support for the **start** RPC operation which programmatically uses the "factory-reset all" or "factory-reset switch all all" for stacks

- The feature is supported when the device is installed using install mode and not when using the legacy bundle mode. During the factory reset operation the current operating system image files are used to boot the device with after the operation completes

- Secure write of the disks with zero's is an optional and supported security feature when the "zero_fill" option and configuration flag is used
  - Zero fill supported on the following platforms: C9300, C9400, C9500, C9500/H*, C9800

* zero fill on C9500/H post 17.7

gNOI reset.proto - tooling

17.7

https://github.com/google/gnxi/tree/master/gnoi_reset

Similar to previous gNOI implementations the recommended tooling is gnoi_reset from Google's gNxI Github repository

show gnxi state detail

```
GNOI
====

  Cert Management service
  -----------------
    Admin state: Enabled
    Oper status: Up

  OS Image service
  ----------------
    Admin state: Enabled
    Oper status: Up
    Supported: Supported

  Factory Reset service
  ---------------------
    Admin state: Enabled
    Oper status: Up
    Supported: Supported
```

Run

```
./gnoi_reset \
    -target_addr localhost:9339 \
    -target_name target.com \
    -rollback_os \
    -zero_fill \
    -key client.key \
    -cert client.crt \
    -ca ca.crt
```

gNOI Factory Reset Client

A simple shell binary that performs Factory Reset operations against a gNOI target. The target will then enter bootstrapping mode.

gNOI Factory Reset Options

- `-rollback_os` will attempt to roll back the OS to the factory version and reset all certificates on the target.
- `-zero_fill` will attempt to zero fill the Target's persistent storage.

Install

```
go get github.com/google/gnxi/gnoi_reset
go install github.com/google/gnxi/gnoi_reset
```

```
auto@pod24-xelab:~$ go get github.com/google/gnxi/gnoi_reset
auto@pod24-xelab:~$ go install github.com/google/gnxi/gnoi_reset
auto@pod24-xelab:~$ gnoi_reset
F1012 14:44:19.032795 2715507 credentials.go:136] Please provide a -target_name
```

go get github.com/google/gnxi/gnoi_reset
go install github.com/google/gnxi/gnoi_reset
auto@pod24-xelab:~$ gnoi_reset
F1012 14:44:19.032795 2715507] Please provide a -target_name

gNOI reset.proto example – zero fill

17.7

auto@pod24-xelab:~$ ./gnoi_reset -target_addr 10.1.1.5:50052 -target_name c9300 -notls -zero_fill
I1014 11:53:15.248633 2761247 gnoi_reset.go:59] Reset Called Successfully!

```
auto@pod24-xelab:~$ ./gnoi_reset -target_addr 10.1.1.5:50052 -target_name dd -notls -zero_fill
I1014 11:53:15.248633 2761247 gnoi_reset.go:59] Reset Called Successfully!
auto@pod24-xelab:~$
```

```
auto@pod24-xelab: ~ (ssh)
C9300#
Chassis 1 reloading, reason - Factory Reset
Oct 14 11:50:31.175: %PMAN-5-EXITACTION: F0/0: pvp: Process manager is exiting: reload fp action requested
Oct 14 11:50:32.839: %PMAN-5-EXITACTION: R0/0: pvp: Pger is exiting: rp processes exit with reload switch code


Enabling factory reset for this reload cycle
 Switch booted with flash:packages.conf
 Switch booted via packages.conf
FACTORY-RESET-RESTORE-IMAGE Taking backup of flash:packages.conf
FACTORY-RESET-RESTORE-IMAGE Searching for packages.conf on flash
factory-reset-restore-image taking a backup of packages.conf from flash
factory-reset-restore-image copying cat9k-cc_srdriver.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-espbase.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-guestshell.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-rpbase.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-sipbase.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-sipspa.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-srdriver.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-webui.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-wlc.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying cat9k-rpboot.BLD_V177_THROTTLE_LATEST_20210929_030812_V17_7_0_106_2.SSA.pkg to /tmp/factory_reset
factory-reset-restore-image copying packages.conf to /tmp/factory_reset
% FACTORYRESET - Started Cleaning Up...

% FACTORYRESET - Unmounting sd1
% FACTORYRESET - Cleaning Up sd1 [0]
% FACTORYRESET - erase In progress.. please wait for completion...
% FACTORYRESET - write zero...
% FACTORYRESET - finish erase

% FACTORYRESET - Making File System sd1 [0]
```

gRPC Tunnel

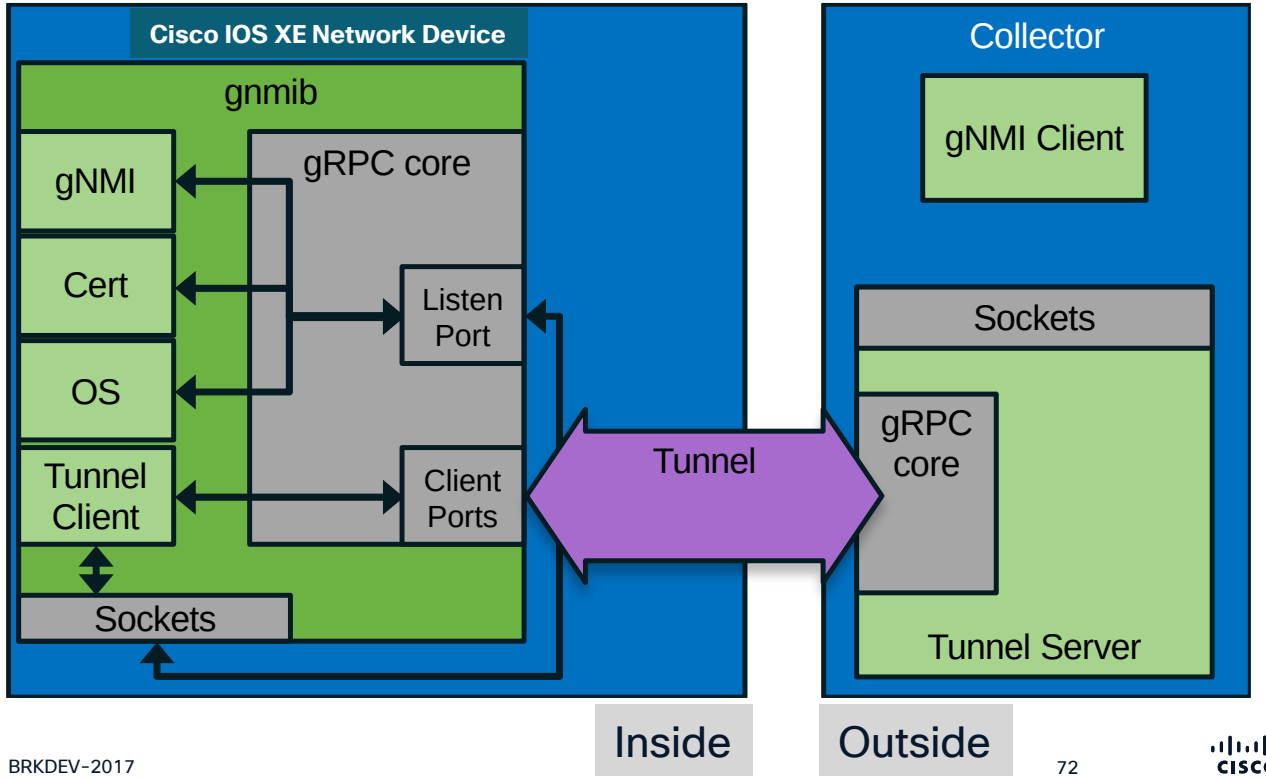
gNMI tunnel.proto

“grpctunnel is an implementation of a TCP-over-gRPC tunnel”

gRPC tunnel

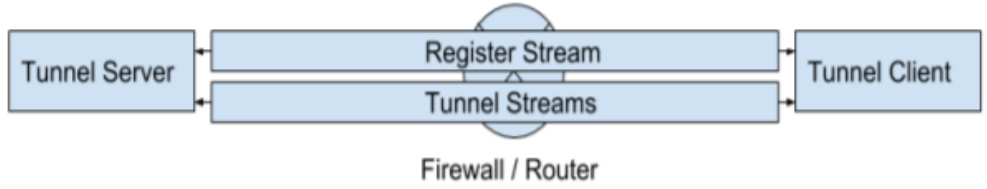
"grpctunnel is an implementation of a TCP-over-gRPC tunnel"
It is very similar to the commonly used "SSH tunnel" concept
https://github.com/openconfig/grpctunnel

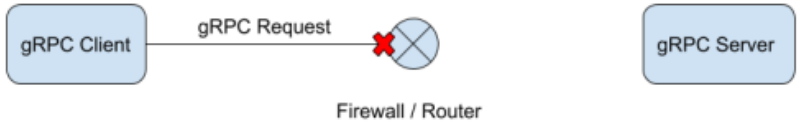
- The devices makes a **secure** outbound connection to the gRPC tunnel server in order to expose the gNMI API for operational use
- Many devices can connect into a single tunnel server in order to increase operational efficiency
- Tunnels can be opened to any number of servers as needed and is not limited to a single tunnel

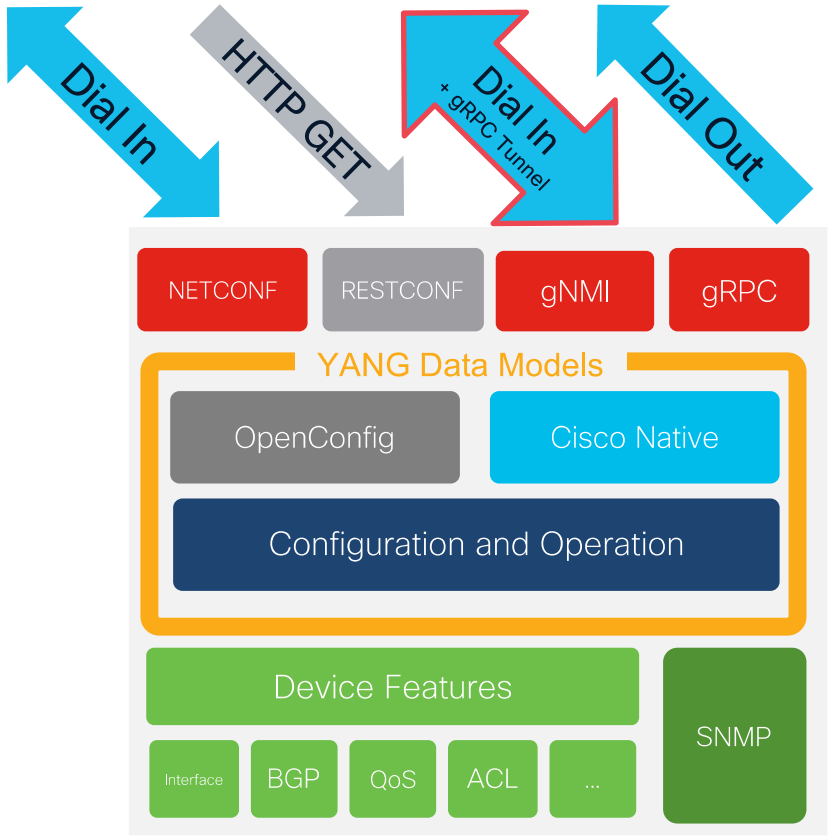


gRPC tunnel

17.11







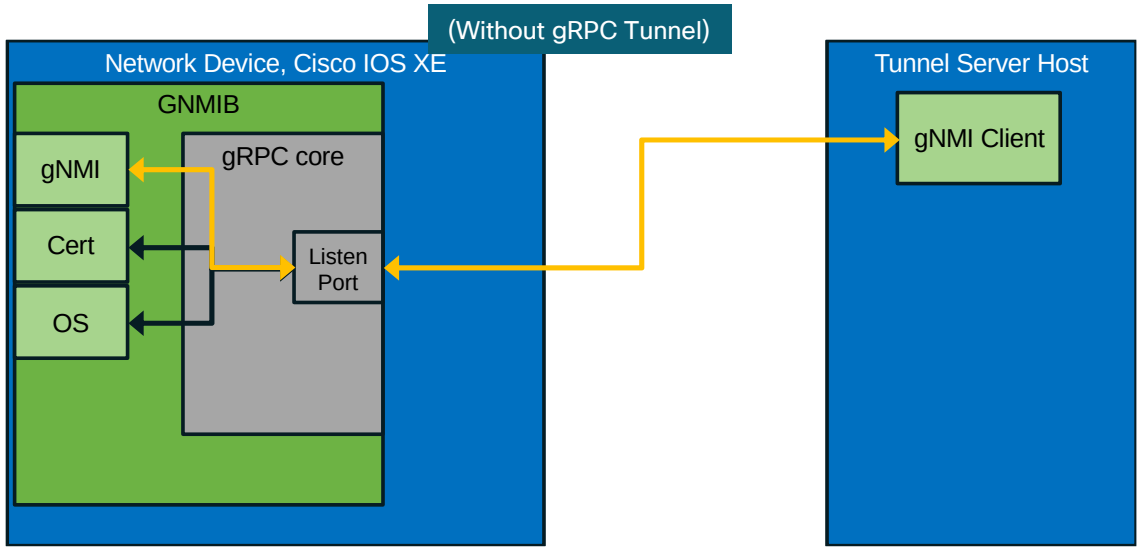
gRPC Dial-Out has seen wide adoption because of the dial-out architecture. gNMI Subscribe has advantages and now also supports Dial-Out with the “grpc tunnel” proto. The grpctunnel tooling is an implementation of a TCP-over-gRPC tunnel, written in Go.

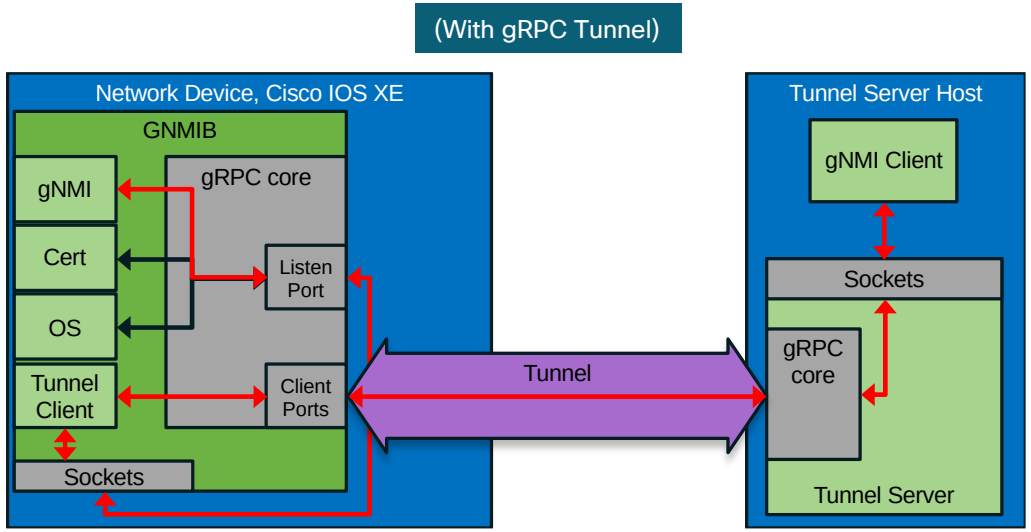
Customer request: “to create a transparent, bi-directional TCP-over-gRPC tunnel supporting gNMI services and other potential protocols such as gNOI in the future”

gNMI Subscribe is sent within the gNMI tunnel to the device as well as all other operations including GET / SET and gNOI proto for certificate, reset, and operating system management

https://github.com/openconfig/grpctunnel/blob/master/proto/tunnel/tunnel.proto
https://github.com/openconfig/grpctunnel

gRPC Tunnel





Without gRPC Tunnel, the gNMI client must connect directly into the Cisco IOS XE gNMI listening port.

With gRPC Tunnel the Cisco IOS XE device will create a **secure** outbound connection to the Tunnel Server Host. From there, the gNMI client can connect into the Cisco IOS XE device within the tunnel.

Q: What is the gNMI Client:
A: YANG Suite's gNMI plugin can be used as the gNMI client for GET/SET/Subscribe – however YANG Suite does not yet support gRPC Tunnel Server

Q: What is the gNMI Tunnel Server Host software?
A: Currently gNMIc is the recommended tooling for tunnel server testing and validation

TLS + mTLS support

https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1710/b_1710_programmability_cg/m_1710_prog_gnmi.html

gRPC Tunnel quick start - configuration

1. Install gnmic tooling
2. Configure gnmic for tunnel server
3. Enable the gnmic tunnel server
4. Enable gNMI API on IOS XE 17.11+
5. Configure and enable gRPC tunnel

```
#1
bash -c "$ (curl -sL https://get-gnmic.openconfig.net)"
```

```
#2
! tunnel_server_config.yaml

insecure: false
skip-verify: true
log: true
username: admin
password: put_yours_here

tunnel-server:
  address: ":4000"
  target-wait-time: "10s"
```

```
#3
gnmic --config ./tunnel_server_config.yaml
--use-tunnel-server subscribe
--path system/config/hostname -i 10s
--stream-mode sample
```

```
#4 conf t
gnxi
gnxi secure-init
service internal
gnxi secure-allow-self-signed-trustpoint
```

```
#5
gnxi grpctunnel dest ubuntuvm
 address 10.1.1.3
 port 4000
!source-vrf Mgmt-vrf
 enable
gnxi grpctunnel target GNMI_GNOI
 enable
```

Match the port 4000 between the tunnel-server and the grpctunnel config
Update the IP, VRF, and credentials as needed for the environment
Secure gNXI configuration example included with self-signed certificates that are not validated

gRPC Tunnel quick start - example

1. Install gnmic as tunnel server
2. Configure gnmic for tunnel service
3. Enable the gnmic server
4. Enable gNMI API
5. Configured and enable gRPC tunnel

```
auto@pod27-xelab:~$
auto@pod27-xelab:~$ bash -c "$(curl -sL https://get-gnmic.openconfig.net)"
gnmic 0.28.0 is available. Changing from version 0.26.0.
Downloading https://github.com/openconfig/gnmic/releases/download/v0.28.0/gnmic_0.28.0_linux_x86_64.tar.gz
Preparing to install gnmic 0.28.0 into /usr/local/bin
gnmic installed into /usr/local/bin/gnmic
version : 0.28.0
 commit : 8315400
   date : 2022-12-07T17:02:16Z
 gitURL : https://github.com/openconfig/gnmic
   docs : https://gnmic.openconfig.net
auto@pod27-xelab:~$
```

#1

```
auto@pod27-xelab:~$
auto@pod27-xelab:~$ cat gnmic/tunnel_server_config.yaml
insecure: false
skip-verify: true
log: true
username: admin
password: Cisco123

tunnel-server:
  address: ":4000"
  target-wait-time: "10s"

auto@pod27-xelab:~$
```

#2

```
auto@pod27-xelab:~/gnmic$ gnmic --config ./tunnel_server_config.yaml --use-tunnel-server sub
scribe --path system/config/hostname -i 10s --stream-mode sample
2023/01/31 09:58:41.273538 [gnmic] version=0.28.0, commit=8315400, date=2022-12-07T1    Z
, gitURL=https://github.com/openconfig/gnmic, docs=https://gnmic.openconfig.net
2023/01/31 09:58:41.273576 [gnmic] using config file "./tunnel_server_config.yaml"
```

#3

```
c9300-pod27#
c9300-pod27#sh run | s gnxi
c9300-pod27#
c9300-pod27#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
c9300-pod27(config)#gnxi
c9300-pod27(config)#gnxi secure-init
c9300-pod27(config)#service internal
c9300-pod27(config)#gnxi secure-allow-self-signed-trustpoint
c9300-pod27(config)#end
c9300-pod27#
c9300-pod27#
c9300-pod27#
c9300-pod27#sh run | s gnxi
gnxi
gnxi secure-allow-self-signed-trustpoint
gnxi secure-trustpoint TP-self-signed-1365501949
gnxi secure-server
c9300-pod27#
```

#4

```
c9300-pod27#
c9300-pod27#sh run | s gnxi
gnxi
gnxi secure-allow-self-signed-trustpoint
gnxi secure-trustpoint TP-self-signed-1365501949
gnxi secure-server
c9300-pod27#
c9300-pod27#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
c9300-pod27(config)#
c9300-pod27(config)#gnxi grpctunnel dest ubuntuvm
c9300-pod27(config-destination)# address 10.1.1.3
c9300-pod27(config-destination)# port 4000
c9300-pod27(config-destination)# !source-vrf Mgmt-vrf
c9300-pod27(config-destination)# enable
c9300-pod27(config-destination)#gnxi grpctunnel target GNMI_GNOI
c9300-pod27(config-target)# enable
c9300-pod27(config-target)#
c9300-pod27(config-target)#end
c9300-pod27#
```

#5

cisco

gNMIc as Tunnel Server tooling - example

```
gnmic \
--config ./tunnel_server_config.yaml  \
--use-tunnel-server subscribe  \
--path system/config/hostname \
-i 10s \
--stream-mode sample
```

```
! tunnel_server_config.yaml

insecure: false
skip-verify: true
log: true
username: admin
password: put_yours_here

tunnel-server:
  address: ":4000"
  target-wait-time: "10s"
```

```
auto@pod27-xelab:~/gnmic$
auto@pod27-xelab:~/gnmic$
auto@pod27-xelab:~/gnmic$ gnmic --config ./tunnel_server_config.yaml --use-tunnel-server subscribe --path system/config/hostname -i 10s --stream-mode sample
2023/01/31 11:15:49.753057 [gnmic] version=0.28.0, commit=8315400, date=2022-12-07T17:02:16Z, gitURL=https://github.com/openconfig/gnmic, docs=https://gnmic.openconfig.net
2023/01/31 11:15:49.753136 [gnmic] using config file "./tunnel_server_config.yaml"
2023/01/31 11:15:49.755027 [gnmic] starting output type file
2023/01/31 11:15:49.755224 [file_output:default-stdout] initialized file output: {"Cfg":{"FileName":"","FileType":"stdout","Format":"json","Multiline":true,"Indent":"  ","Separator":"\n","OverrideTimestamps":false,"AddTarget":"","TargetTemplate":"","EventProcessors":null,"MsgTemplate":"","ConcurrencyLimit":1000,"EnableMetrics":false,"Debug":false}}
2023/01/31 11:16:23.220697 [gnmic] tunnel server discovered target {ID:gnmib tunnel test client Type:GNMI_GNOI}
2023/01/31 11:16:23.220871 [gnmic] adding target {"name":"gnmib tunnel test client","address":"gnmib tunnel test client","username":"admin","password":"****","timeout":10000000000,"insecure":false,"tls-cert":"","tls-key":"","skip-verify":true,"buffer-size":100,"retry-timer":10000000000,"log-tls-secret":false,"gzip":false,"token":"","tunnel-target-type":"GNMI_GNOI"}
2023/01/31 11:16:23.220912 [gnmic] queuing target "gnmib tunnel test client"
2023/01/31 11:16:23.220927 [gnmic] starting target "gnmib tunnel test client" listener
2023/01/31 11:16:23.220950 [gnmic] subscribing to target: "gnmib tunnel test client"
2023/01/31 11:16:23.221302 [gnmic] dialing tunnel connection for tunnel target "gnmib tunnel test client"
2023/01/31 11:16:23.248063 [gnmic] target "gnmib tunnel test client" gNMI client created
2023/01/31 11:16:23.248468 [gnmic] sending gNMI SubscribeRequest: subscribe='subscribe:{subscription:{path:{elem:{name:"system"} elem:{name:"config"} elem:{name:"hostname"}} mode:SAMPLE sample_interval:10000000000}}', mode='STREAM', encoding='JSON', to gnmib tunnel test client
{
  "source": "gnmib tunnel test client",
  "subscription-name": "default-1675192549",
  "timestamp": 1675192583304250000,
  "time": "2023-01-31T11:16:23.30425-08:00",
  "updates": [
    {
      "Path": "system/config/hostname",
      "values": {
        "system/config/hostname": "c9300-pod27"
      }
    }
  ]
}
```

gRPC Tunnel quick start - validation

show run | s gnxi
sh gnxi grpc dest

```
c9300-pod27#sh run | s gnxi
gnxi
gnxi secure-allow-self-signed-trustpoint
gnxi secure-trustpoint TP-self-signed-1365501949
gnxi secure-server
gnxi grpctunnel destination ubuntuvm
 address 10.1.1.3
 port 4000
 enable
gnxi grpctunnel target GNMI_GNOI
 enable
c9300-pod27#
```

```
c9300-pod27#
c9300-pod27#show gnxi grpctunnel destinations
All configured destinations

Destination Name: ubuntuvm
  Target: GNMI_GNOI
  Tag: 1
  Registered: Yes
  Session Started: Yes
  Tunnel Active: Yes
  Error:


c9300-pod27#
```

```
auto@pod27-xelab:~/gnmic$
auto@pod27-xelab:~/gnmic$ gnmic --config ./tunnel_server_config.yaml
2023/01/31 11:15:49.753057 [gnmic] version=0.28.0, commit=8315400, da
.net
2023/01/31 11:15:49.753136 [gnmic] using config file "./tunnel_server
2023/01/31 11:15:49.755027 [gnmic] starting output type file
2023/01/31 11:15:49.755224 [file_output:default-stdout] initialized f
 ","Separator":"\n","OverrideTimestamps":false,"AddTarget":"","Target
,"Debug":false}}
2023/01/31 11:16:23.220697 [gnmic] tunnel server discovered target {I
2023/01/31 11:16:23.220871 [gnmic] adding target {"name":"gnmib tunne
:10000000000,"insecure":false,"tls-cert":"","tls-key":"","skip-verify
"tunnel-target-type":"GNMI_GNOI"}
2023/01/31 11:16:23.220912 [gnmic] queuing target "gnmib tunnel test
2023/01/31 11:16:23.220927 [gnmic] starting target "gnmib tunnel test
2023/01/31 11:16:23.220950 [gnmic] subscribing to target: "gnmib tunn
2023/01/31 11:16:23.221302 [gnmic] dialing tunnel connection for tunn
2023/01/31 11:16:23.248063 [gnmic] target "gnmib tunnel test client"
2023/01/31 11:16:23.248468 [gnmic] sending gNMI SubscribeRequest: sub
ame"}} mode:SAMPLE sample_interval:10000000000}}', mode='STREAM', enc
{
  "source": "gnmib tunnel test client",
  "subscription-name": "default-1675192549",
  "timestamp": 1675192583304250000,
  "time": "2023-01-31T11:16:23.30425-08:00",
  "updates": [
    {
      "Path": "system/config/hostname",
      "values": {
        "system/config/hostname": "c9300-pod27"
      }
    }
  ]
```

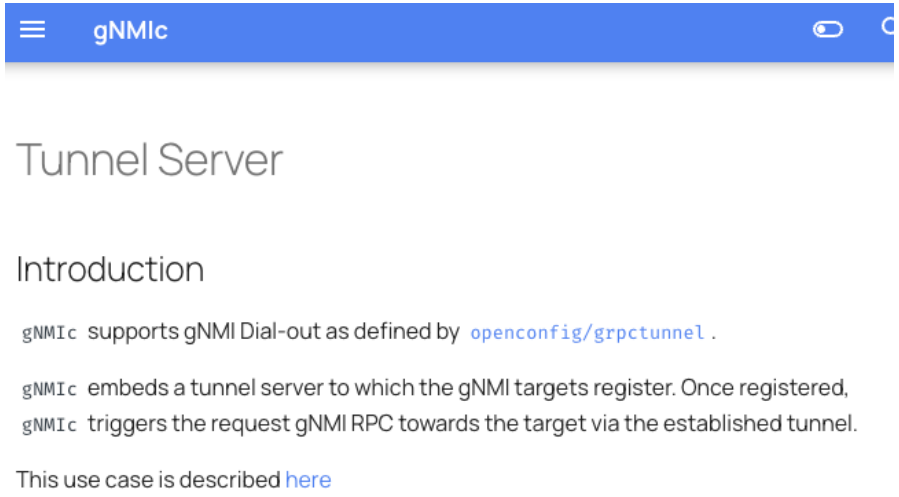
gNMIc as Tunnel Server tooling

gNMIc is common tooling that supports the grpc tunnel
https://gnmic.openconfig.net/user_guide/tunnel_server/

```
auto@pod27-xelab:~$ gnmic version
version : 0.28.0
 commit : 8315400
   date : 2022-12-07T17:02:16Z
 gitURL : https://github.com/openconfig/gnmic
   docs : https://gnmic.openconfig.net
```

Configuration

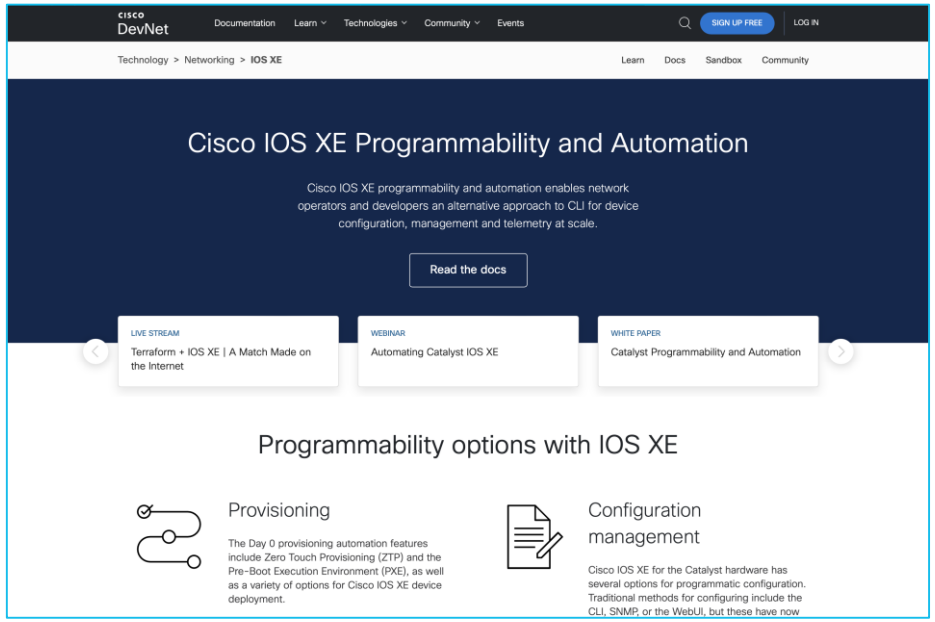
```
tunnel-server:
  # the address the tunnel server will listen to
  address:
  # if true, the server will not verify the client's certificates
  skip-verify: false
  # path to the CA certificate file to be used, irrelevant if `skip-verify` is
  ca-file:
  # path to the server certificate file
  cert-file:
  # path to the server key file
  key-file:
  # the wait time before triggering unary RPCs or subscribe poll/once
  target-wait-time: 2s
  # enables the collection of Prometheus gRPC server metrics
  enable-metrics: false
  # enable additional debug logs
  debug: false
```



Resources

Programmability Website

The one-stop-shop for Cisco IOS XE Programmability resources including videos, white papers, labs and more!



- Community Forum
- IOS XE FAQ
- White Papers
- Code Exchange
- IOS XE Docs & Guide
- Learning Tracks and Labs
- Sandboxes
  - ... and more !

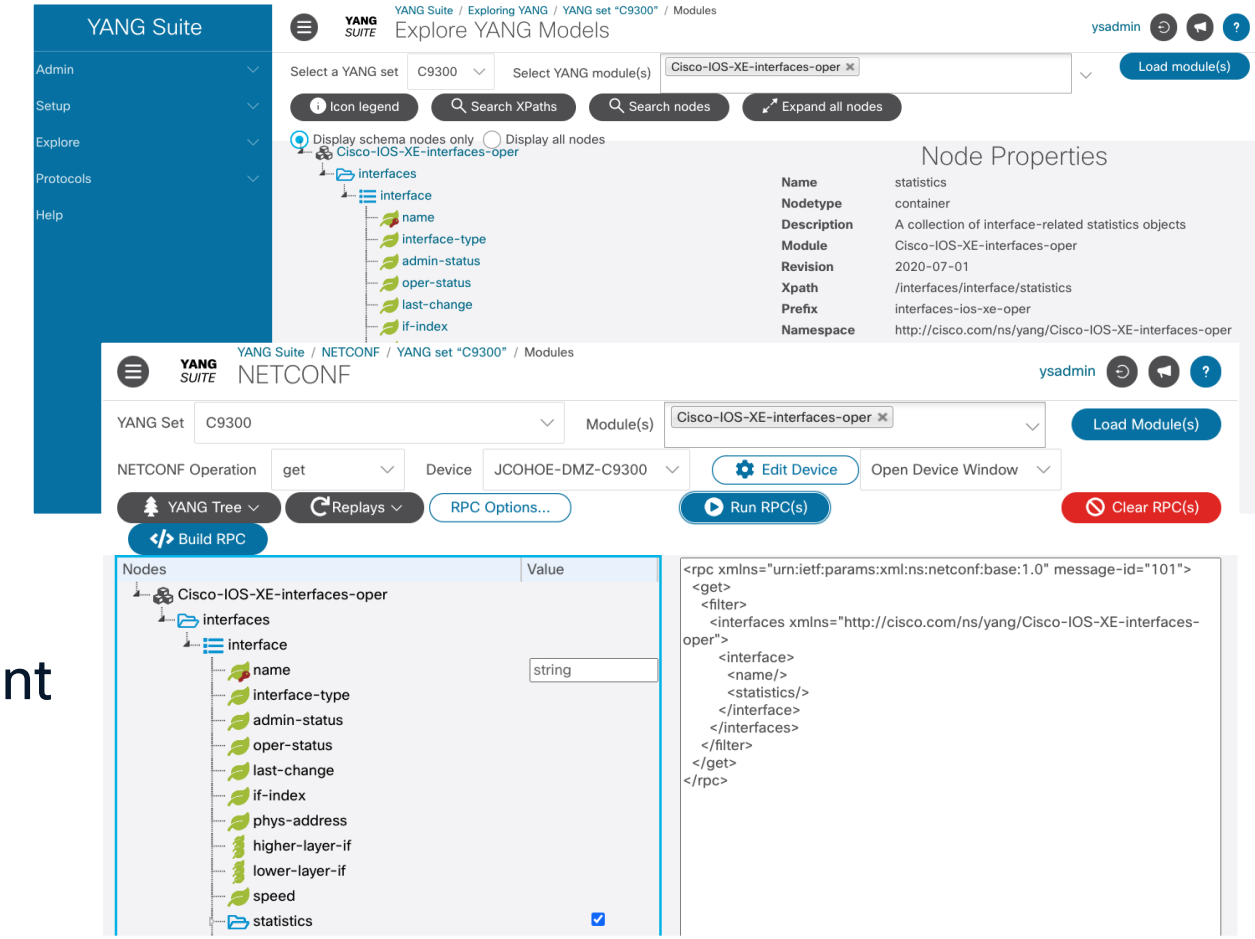
https://developer.cisco.com/iosxe/

Cisco YANG Suite

YANG API Testing and Validation Environment

Construct and test YANG based APIs over NETCONF, RESTCONF, gRPC and gNMI

IOS XE / IOS XR / NX OS platforms



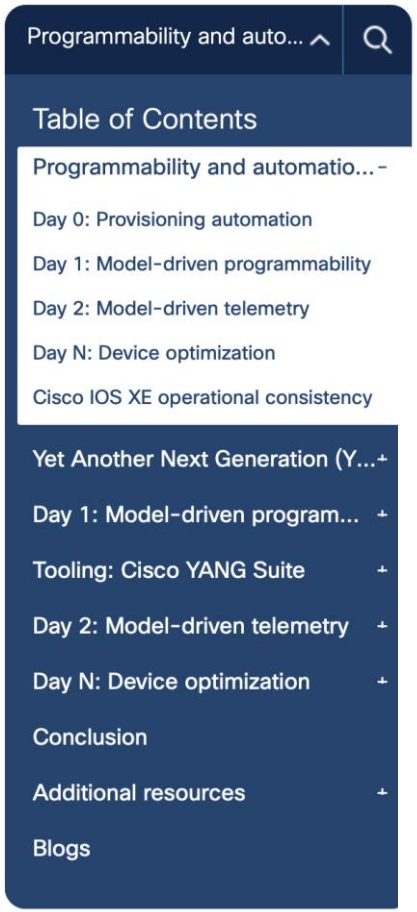
developer.cisco.com/yangsuite

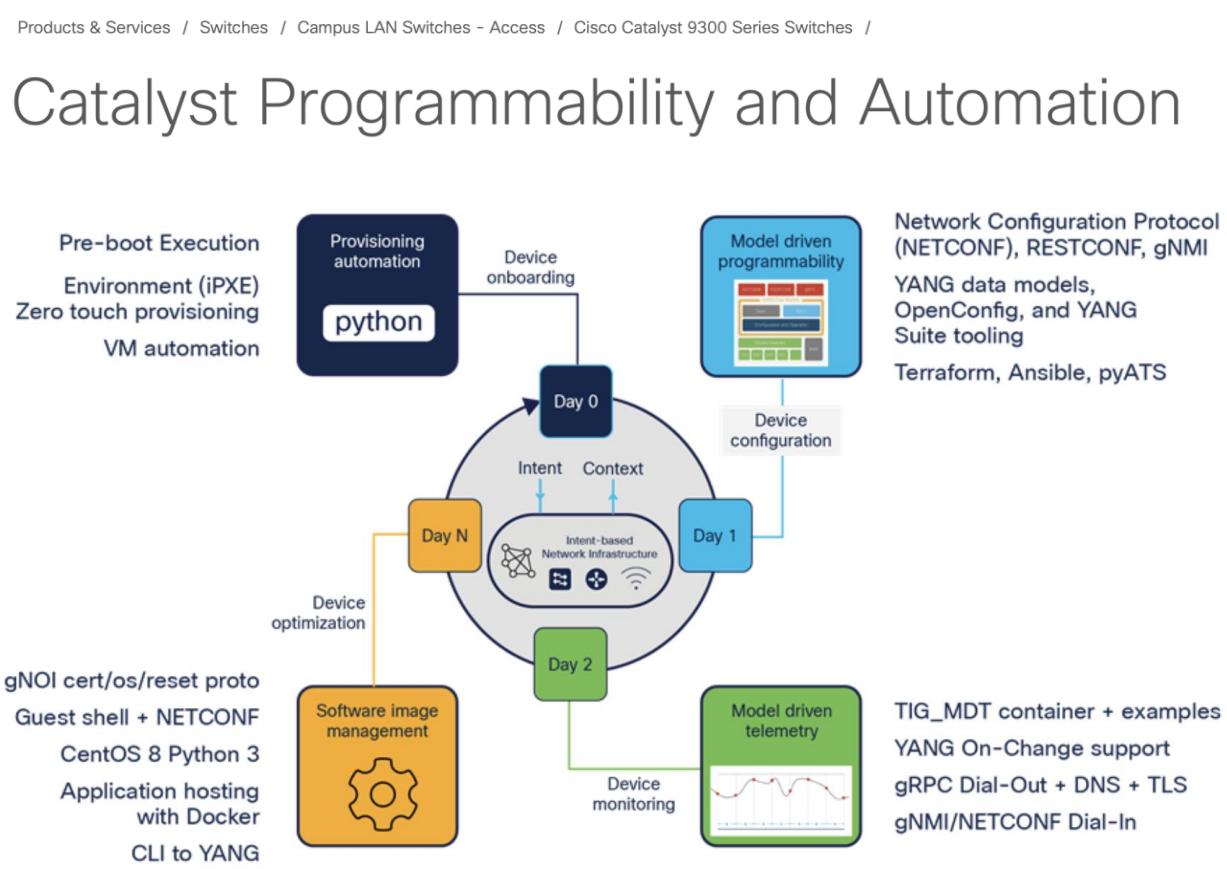
Get hands-on using the new learning lab!
https://developer.cisco.com/learning/labs/intro-yangsuite/
Docker container innovation 1 container

github.com/CiscoDevNet/yangsuite

API White Paper

http://cs.co/apiwp





https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-9300-series-switches/nb-06-catalyst-programmability-automation-wp.html

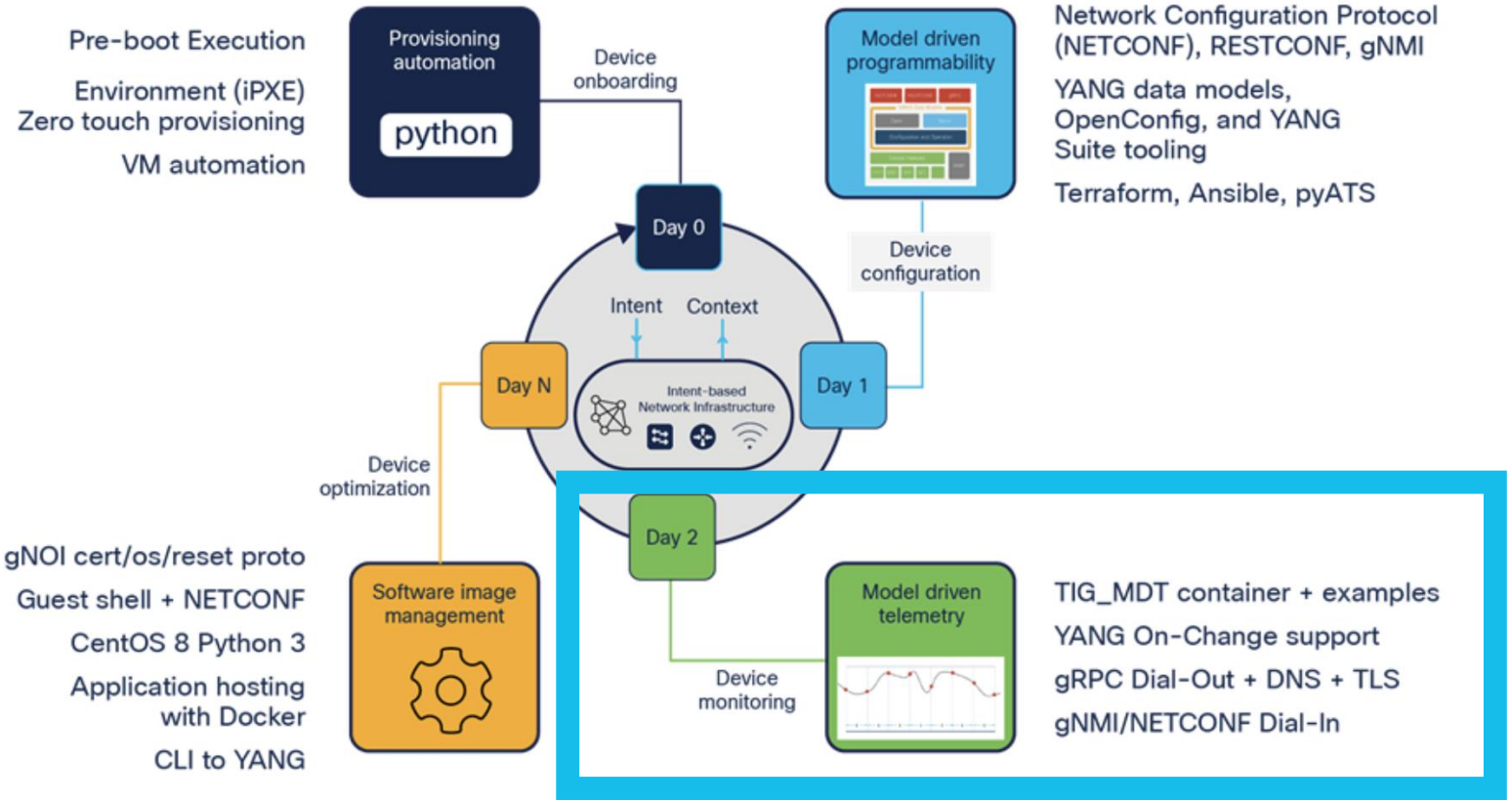
https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-9300-series-switches/nb-06-catalyst-programmability-automation-wp.pdf

https://www.youtube.com/watch?v=LdcK5PnPu2I

http://cs.co/apiwppdf

Model Drive Telemetry (MDT) White Paper

The Model Driven Telemetry White Paper includes examples, use cases and tooling related to telemetry. This paper is now available online and in PDF form!



View online: https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-9300-series-switches/model-driven-telemetry-wp.html
View as PDF: https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-9300-series-switches/model-driven-telemetry-wp.pdf

dCloud Programmability

https://dcloud.cisco.com
"Cisco Catalyst 9000 IOS XE Programmability & Automation Lab v1"
https://dcloud2.cisco.com/demo/catalyst-9000-ios-xe-programmability-automation-lab-v1

Use Cases:

EVPN:

Ansible with CLI deployment of EVPN solutions
EVPN management over RESTCONF/YANG with Postman
Declarative EVPN fabric management with Terraform

Model Driven Telemetry

Telemetry configuration with CLI and YANG Suite
Collection with TIG_MDT container and tooling

YANG Programmability

YANG Suite tooling and integrations to YANG API's
Ansible integrations

Tooling and Integrations

YANG Suite
- NETCONF/RESTCONF/gNMI API
  - Ansible integration
- NETCONF/gNMI Dial-In Telemetry
- gRPC Dial-Out Telemetry receiver

Telemetry
- TIG stack in Docker
- Grafana dashboard for device health

Postman / RESTCONF
- EVPN fabric API calls

Terraform/RESTCONF
- Declarative EVPN fabric management

Ansible
- EVPN solution enablement using CLI

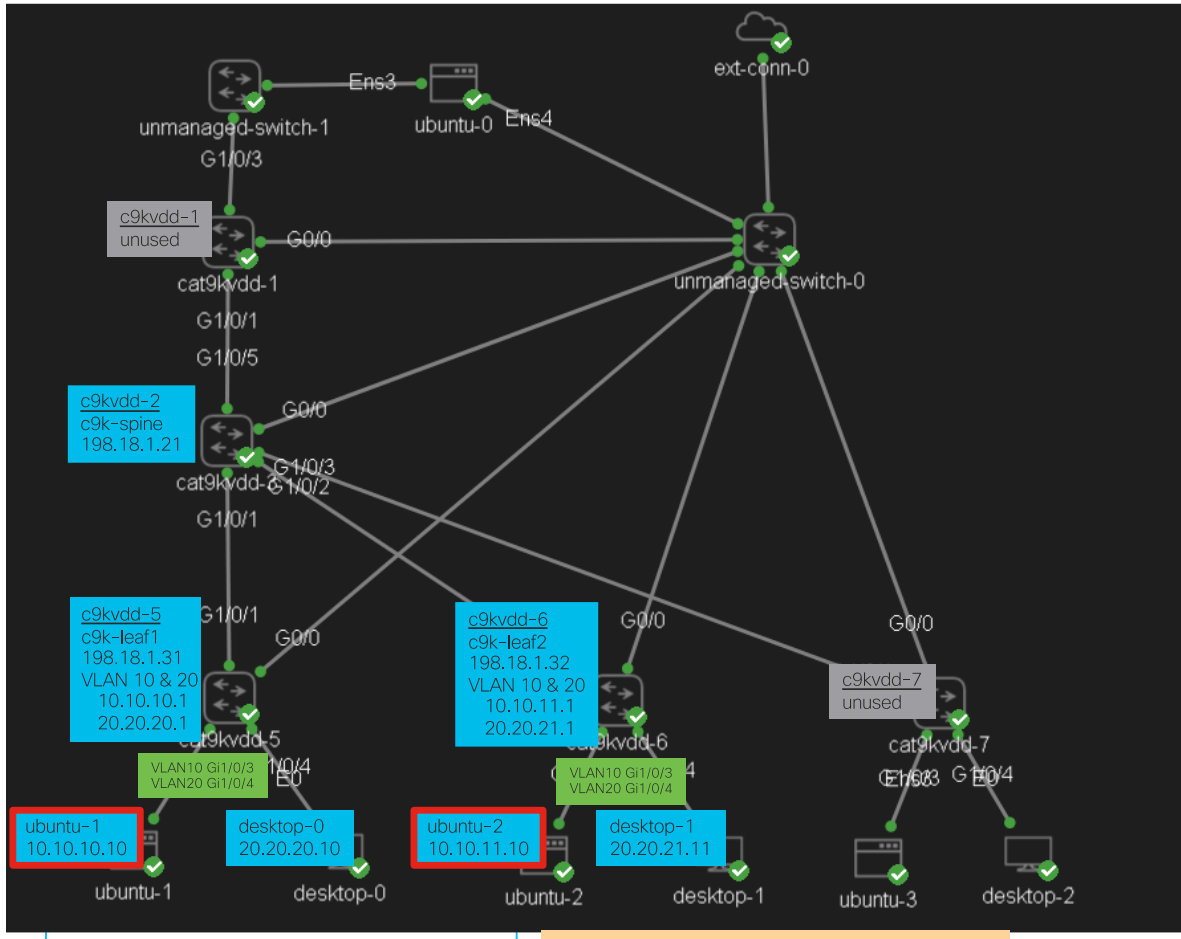
Ubuntu VM Details:

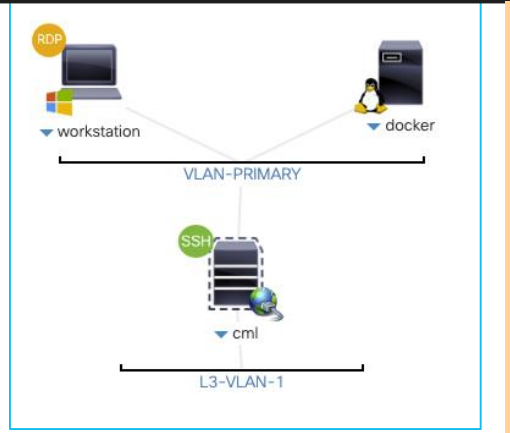
Syslog receiver from all switches
TFTP config backup
See slide

Windows VM Details

VS Code
Terraform @ folder
Ansible @ folder
Chrome browser
YANG Suite, Grafana
Bash/PS/Cmd shells
SSH into C9K or Ubuntu
Postman
Workspace for EVPN

C9K VM's





VLAN1
c9k-spine
IP: 198.18.1.21
developer /
C1sco12345
c9k-leaf1
IP: 198.18.1.31
developer /
C1sco12345
c9k-leaf2
IP: 198.18.1.32
developer /
C1sco12345
c9kvdd-1 - unconfigured

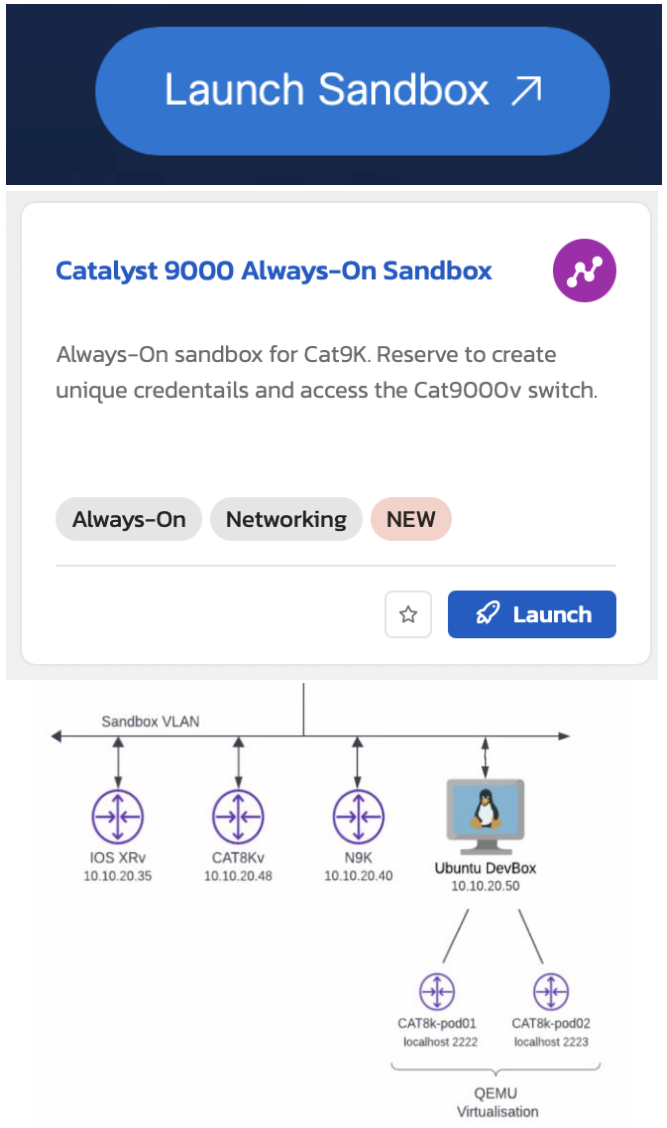
c9kvdd-7 - unconfigured

DevNet Sandbox – overview for Campus and Enterprise

https://developer.cisco.com/site/sandbox/

1. Reservable Physical: C9200, C9300, C9300X including stacks
   About to go into production April 2025
   Usecases: Application Hosting, Power telemetry, etc
2. Reservable Virtual: C8KV Router + NX + XR + Ubuntu VM
   Usecases: Enterprise topology, dual-ZTP
3. Always-On: C9KV
   Usecase: Virtual switch for basic config validation usecases
   DNS: devnetsandboxiosxec9k.cisco.com
   "Launch Sandbox" to get login credentials
4. Always-On: C8KV
   DNS is devnetsandboxiosxe.cisco.com
5. Reservable Catalyst Center (Physical & Virtual)
   CML and C9KV, ISE for SDA

C9KV user is priv15 and has full permissions, there is reset automation for when you break it now too ☺



Additional enablement labs:

1. YANG Suite "Learning Lab 2.0"
   Interactive guide with tool running in Docker container
   "YANG Suite as a service"
2. dCloud Programmability Lab
   EVPN topology with all programmability features enabled

Complete Your Session Evaluations

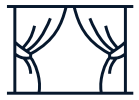
Complete a minimum of 4 session surveys and the Overall Event Survey to be entered in a drawing to win 1 of 5 full conference passes to Cisco Live 2026.

Earn 100 points per survey completed and compete on the Cisco Live Challenge leaderboard.

Level up and earn exclusive prizes!

Complete your surveys in the Cisco Live mobile app.

Continue your education



Visit the Cisco Showcase for related demos

Book your one-on-one Meet the Engineer meeting

Attend the interactive education with DevNet, Capture the Flag, and Walk-in Labs

Visit the On-Demand Library for more sessions at www.CiscoLive.com/on-demand

Thank you

cisco Live !


CISCO