

The background features a vibrant, abstract design with a color gradient from dark blue on the left to bright yellow and white on the right. The design consists of overlapping, wavy horizontal bands and a radial pattern of lines emanating from a bright white point on the right side, creating a sense of motion and energy.

CISCO *Live!*

Let's go



The bridge to possible

# Programmability, Automation Model Driven Telemetry on Cisco IOS XE with a dash of YANG Suite

Story DeWeese, Technical Marketing  
@StoryDeWeese

# Agenda



Intro to Programmability and Automation

Day 0

Day 1

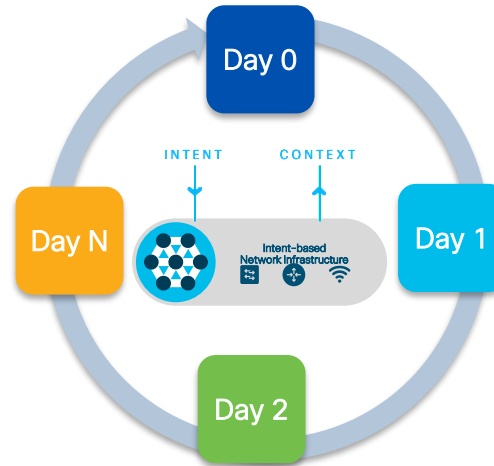
Day 2

Day N

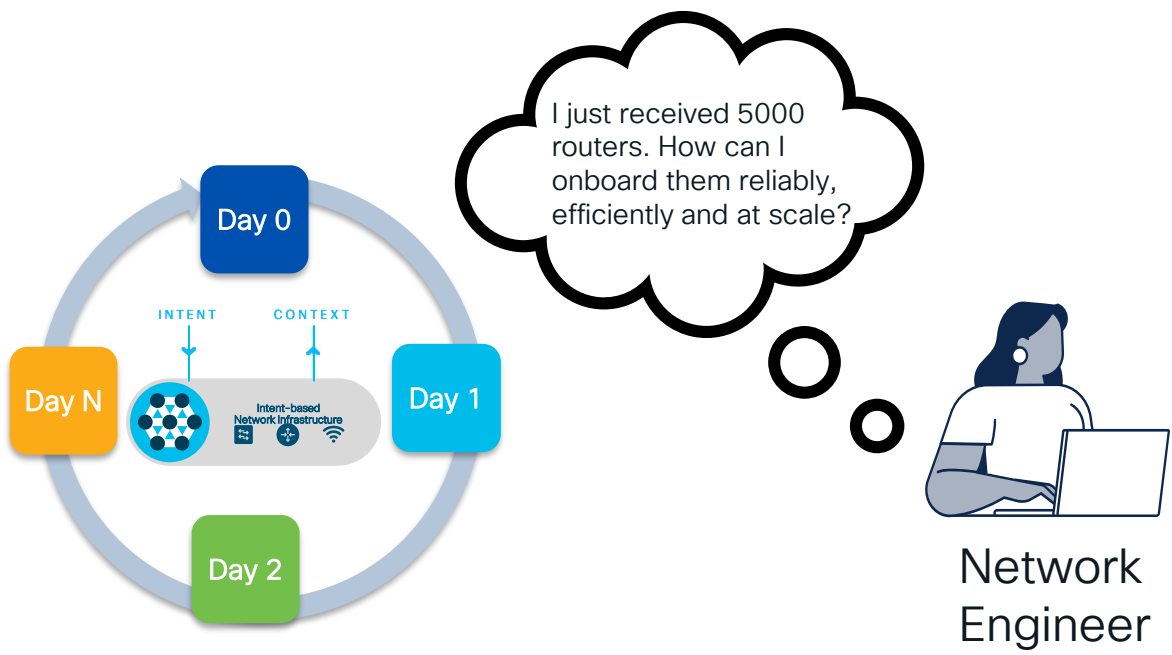
Tooling

Resources

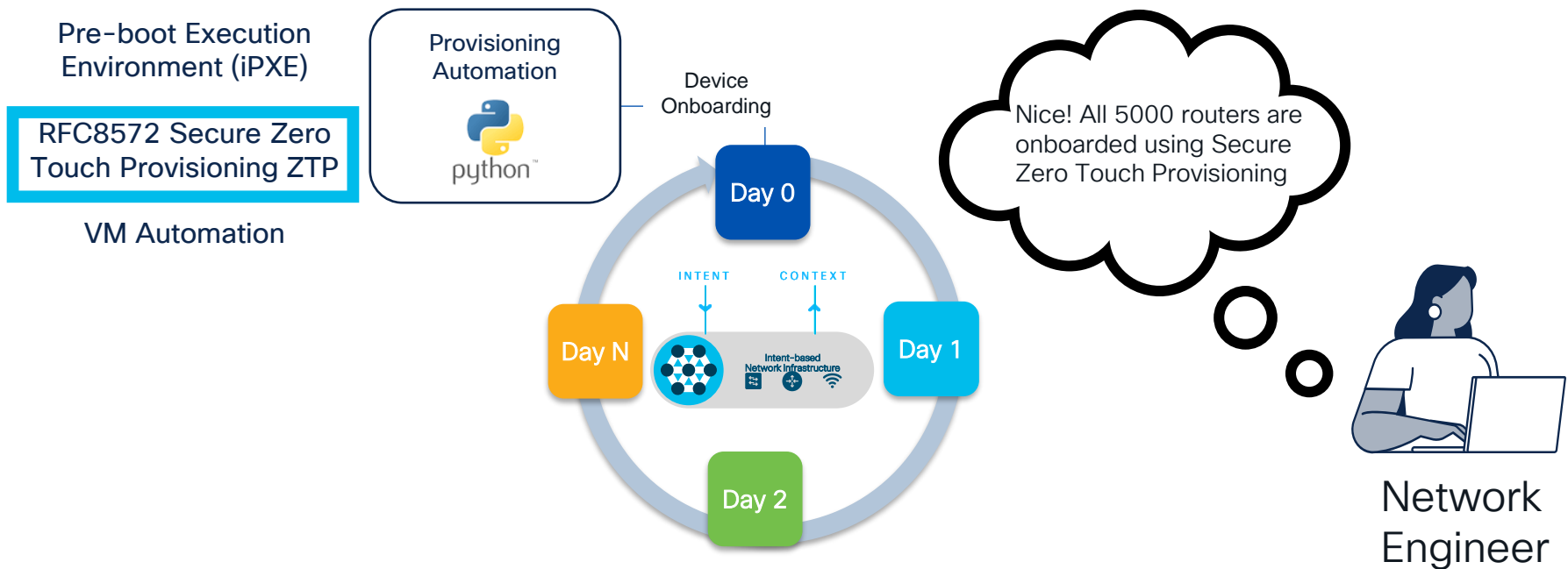
# Cisco IOS XE Programmability & Automation Lifecycle



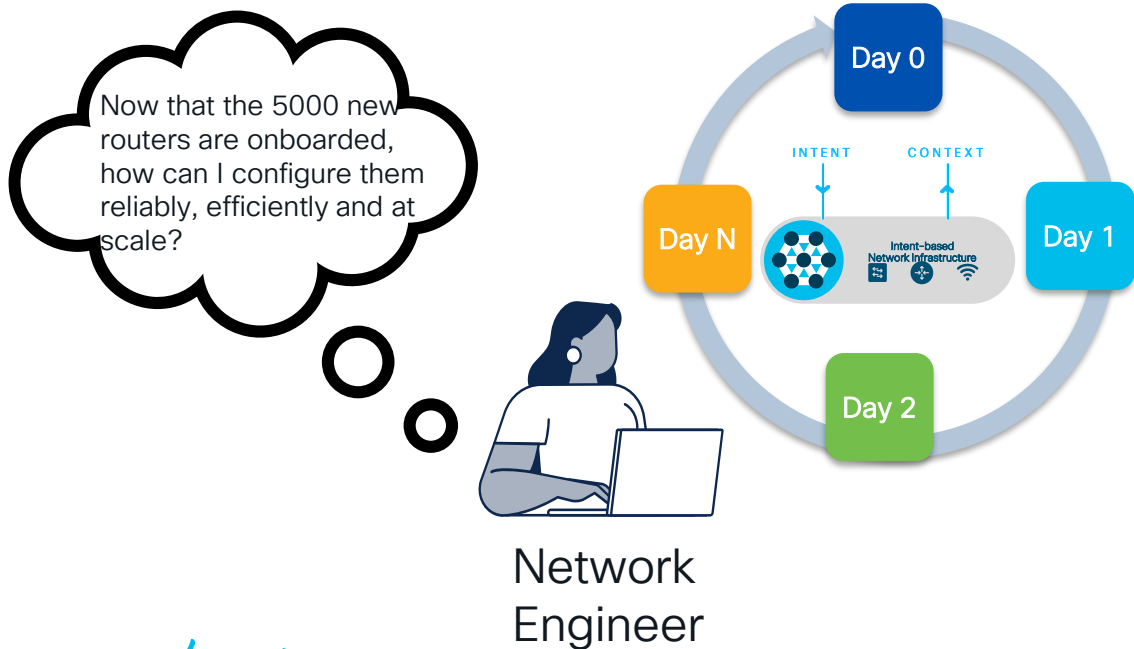
# Cisco IOS XE Programmability & Automation Lifecycle



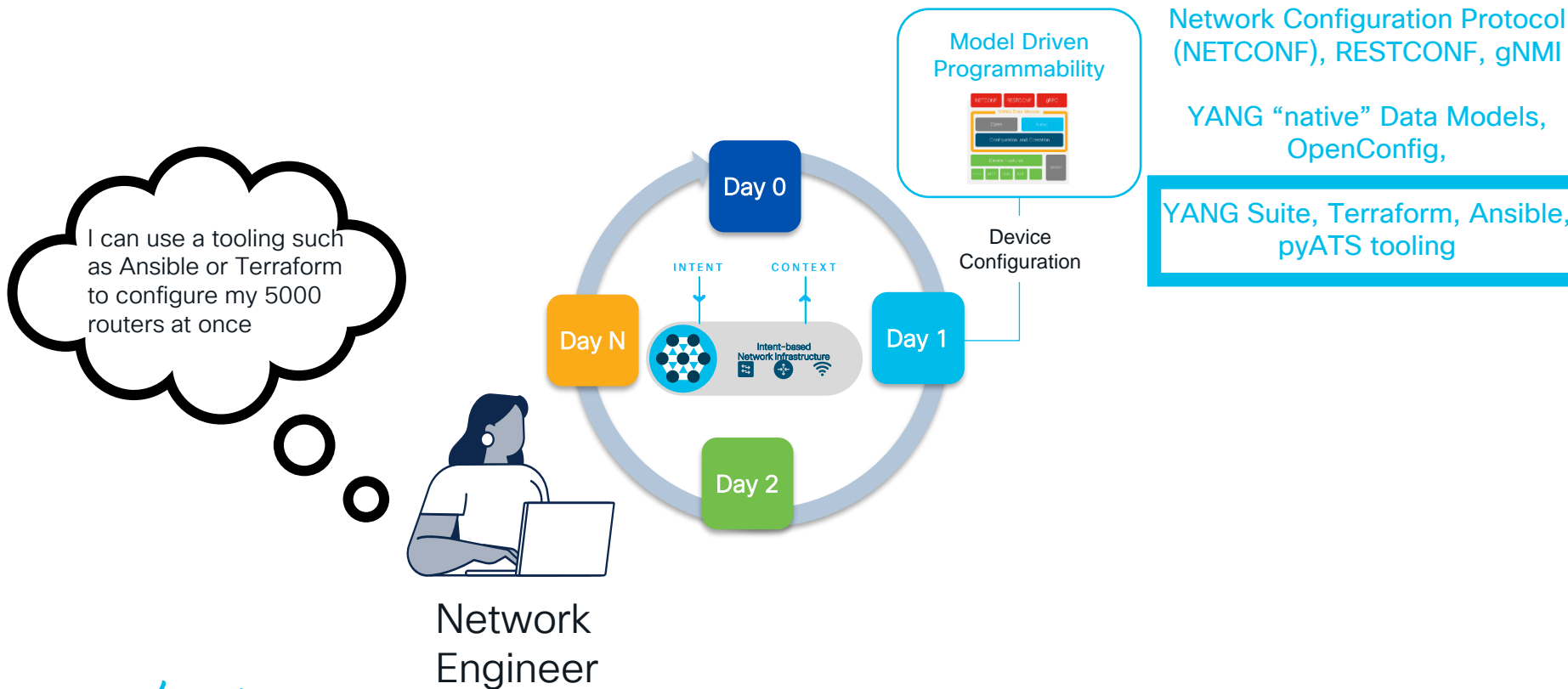
# Cisco IOS XE Programmability & Automation Lifecycle



# Cisco IOS XE Programmability & Automation Lifecycle

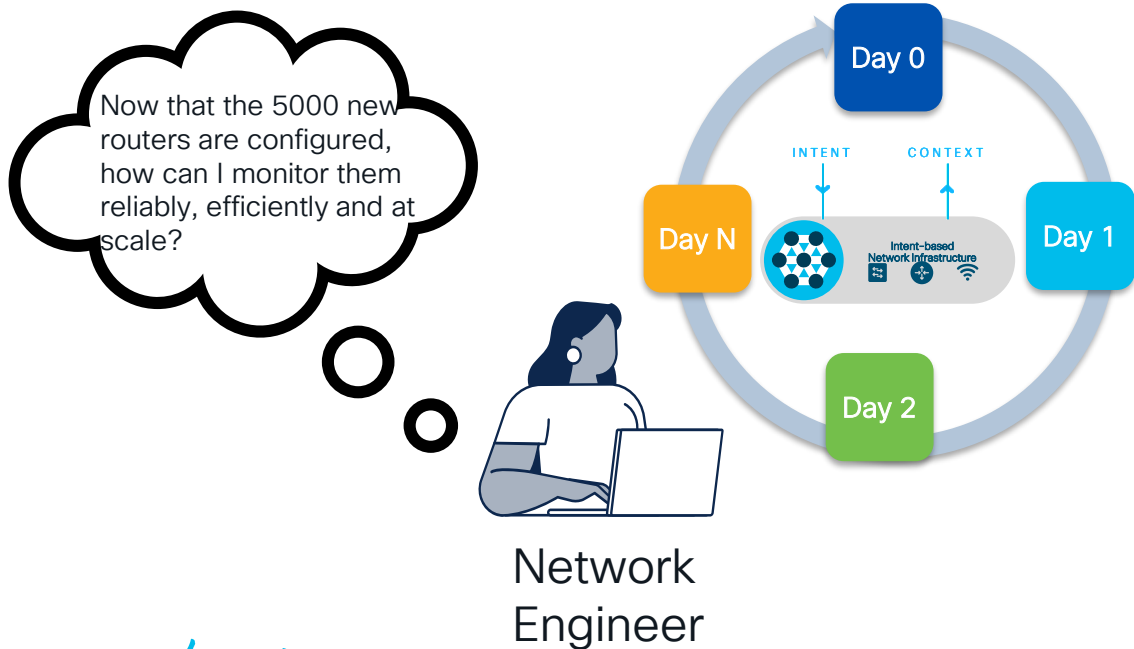


# Cisco IOS XE Programmability & Automation Lifecycle

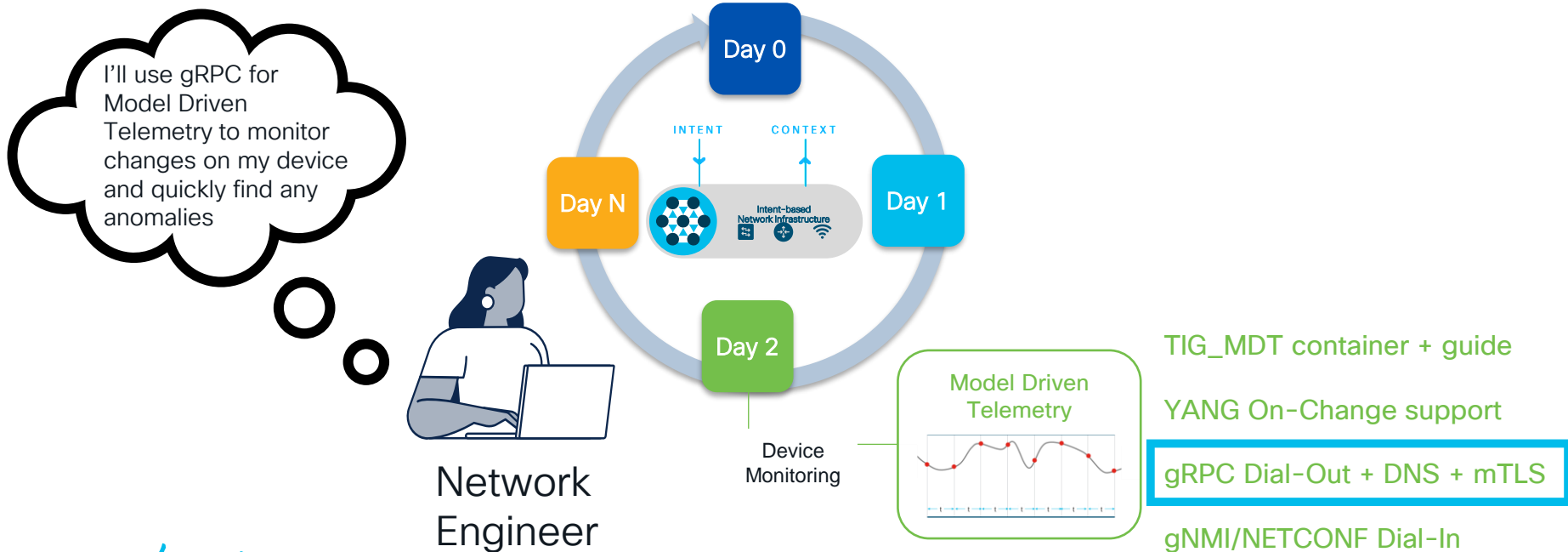




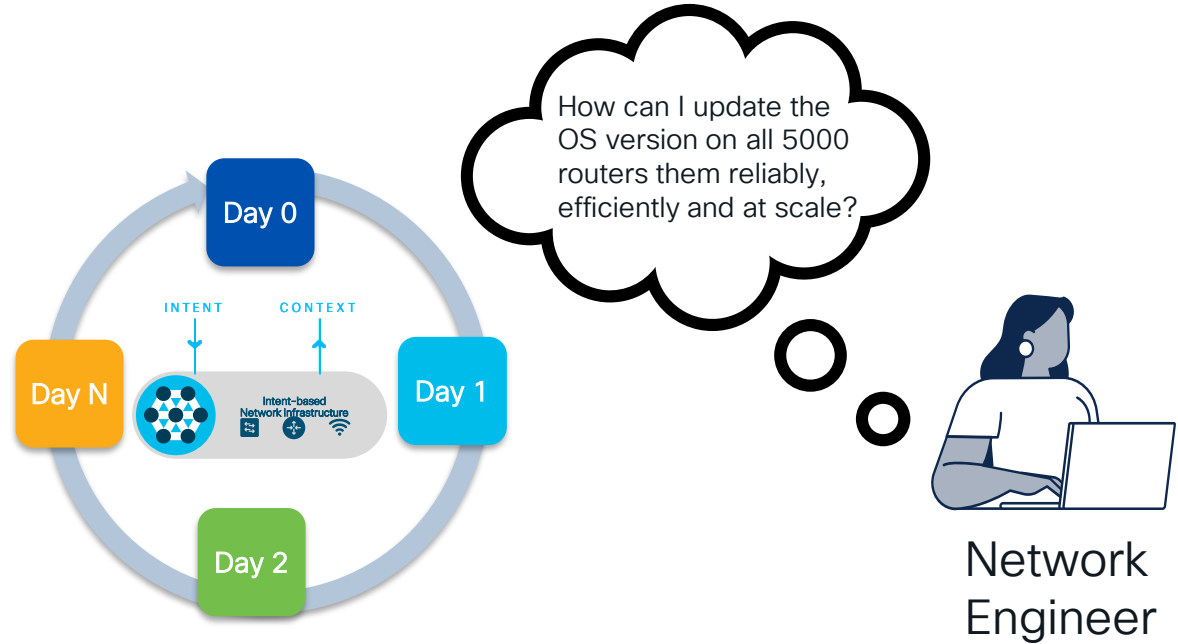
# Cisco IOS XE Programmability & Automation Lifecycle



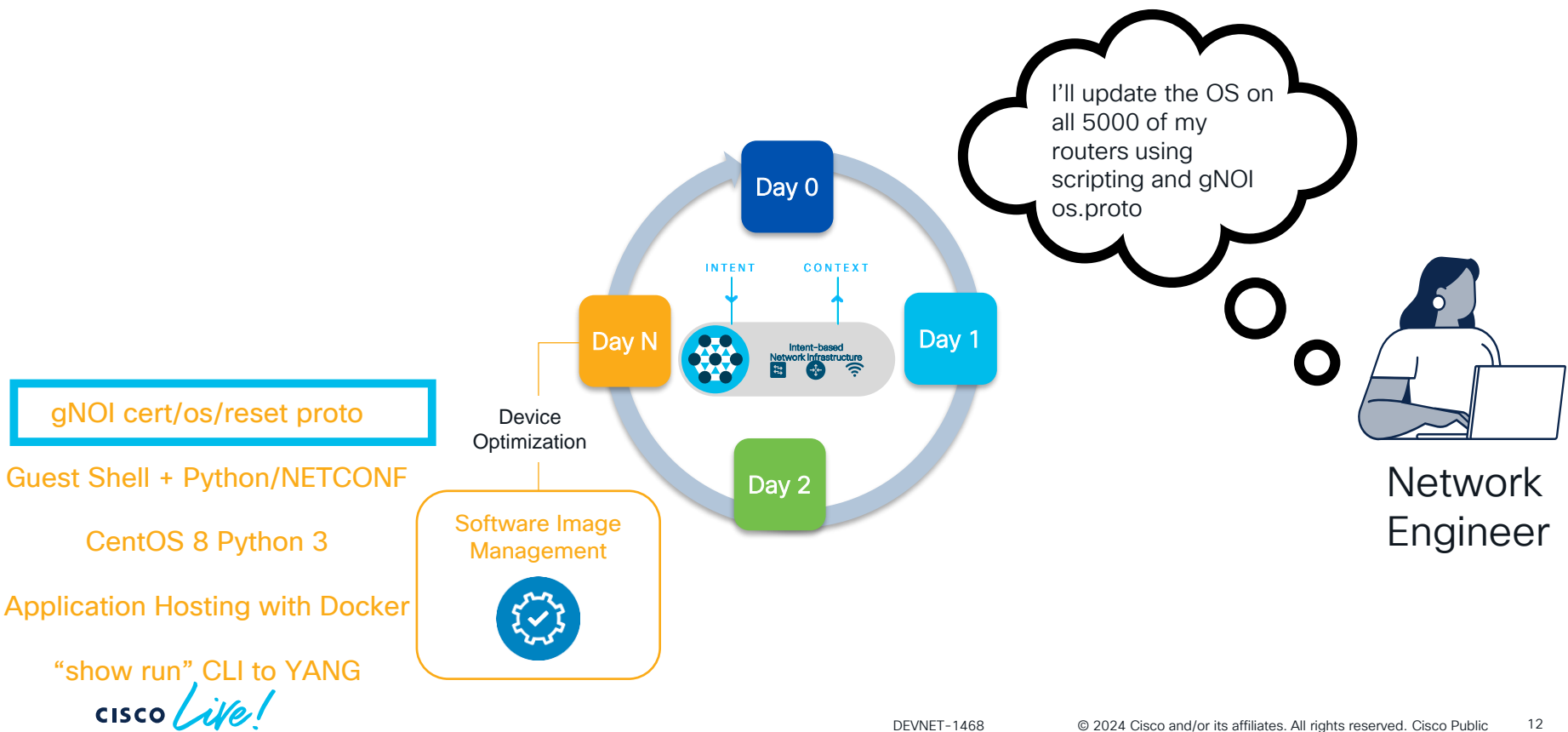
# Cisco IOS XE Programmability & Automation Lifecycle



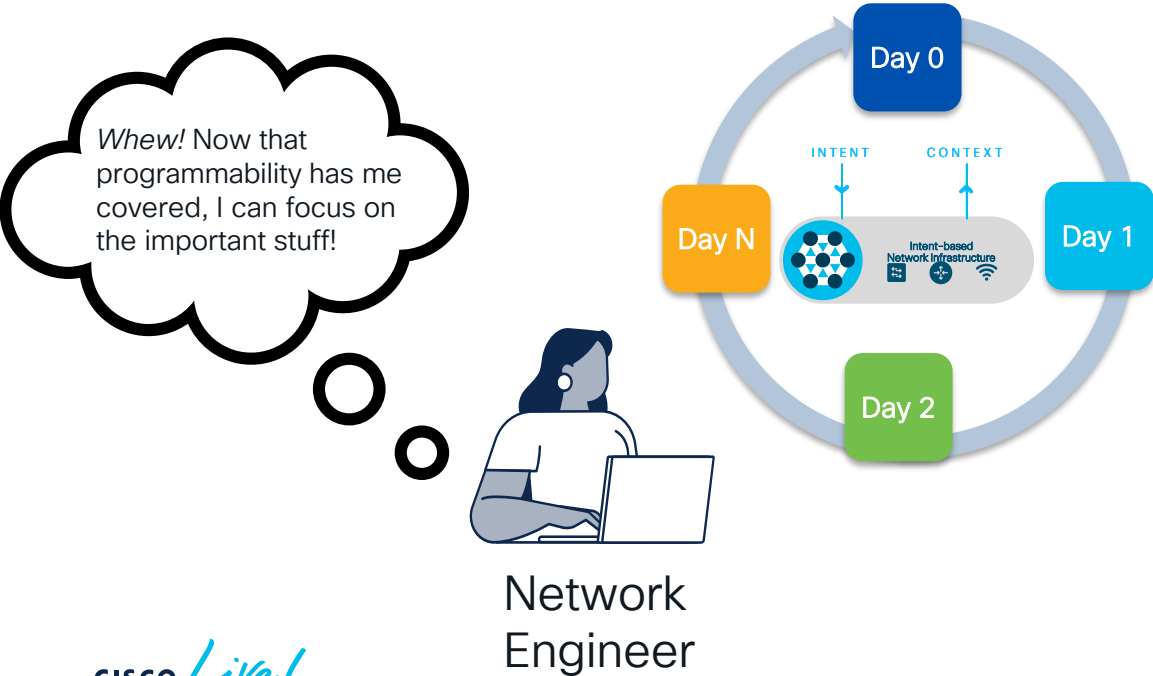
# Cisco IOS XE Programmability & Automation Lifecycle



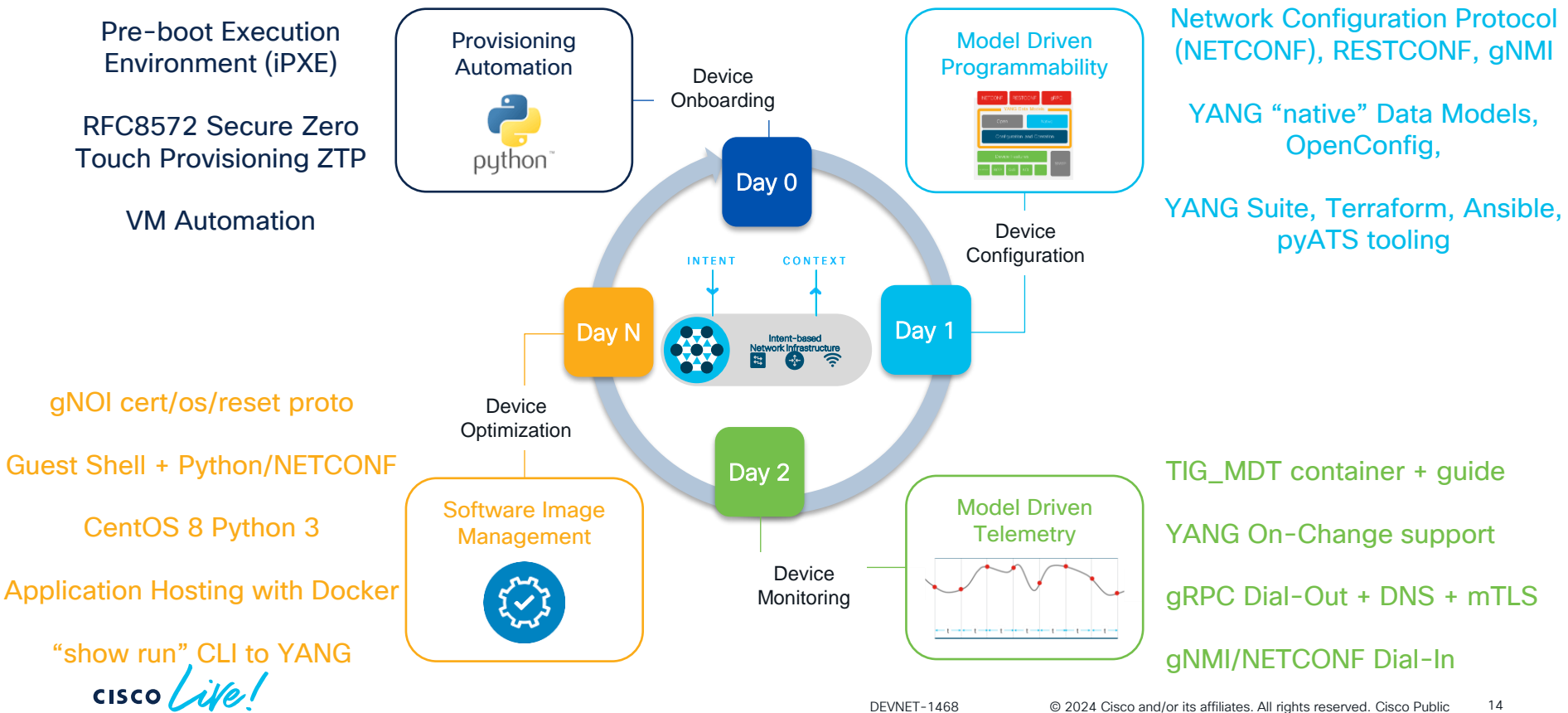
# Cisco IOS XE Programmability & Automation Lifecycle



# Cisco IOS XE Programmability & Automation Lifecycle



# Cisco IOS XE Programmability & Automation Lifecycle



# Model Driven Programmability Interface Comparison

Network architecture, security posture and policy, YANG data modules, tools and language preferences are some considerations when leveraging the various MDP interfaces

	NETCONF	RESTCONF	gNMI
Minimum IOS XE Version	16.6	16.7	16.8
Recommended Version	17.6	17.6	17.7
Default Port	830	443	9339
Operations	<get>,<get-config>,<edit-config>,<establish-subscription>	GET, POST, PUT, PATCH, DELETE	GET, SET, SUBSCRIBE
Encoding	XML	XML or JSON	RFC7951 JSON_IETF
Security	SSH + PKI certificate or password	HTTPS user/pass	TLS certificate with user authentication
Transport Protocol	SSH	HTTPS	HTTP/2
Tooling	YANG Suite, ncclient, Netconf-console	YANG Suite*, Postman, python	YANG Suite*, gnmic, gnmi_cli
Content	YANG	YANG	YANG + Protobuf

# IOS XE Programmability and Telemetry “Stack”

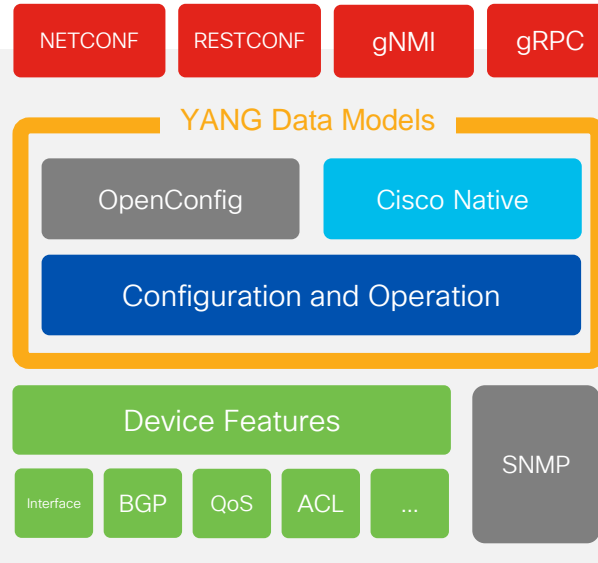
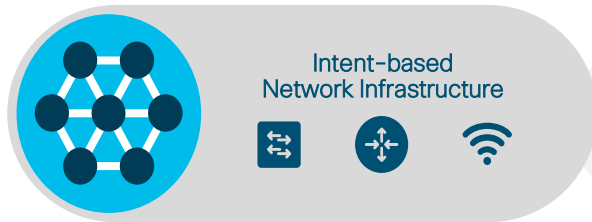
CLI

SNMP

WebUI

The NETCONF, RESTCONF, gNMI and gRPC 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, gNMI, and gRPC.

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



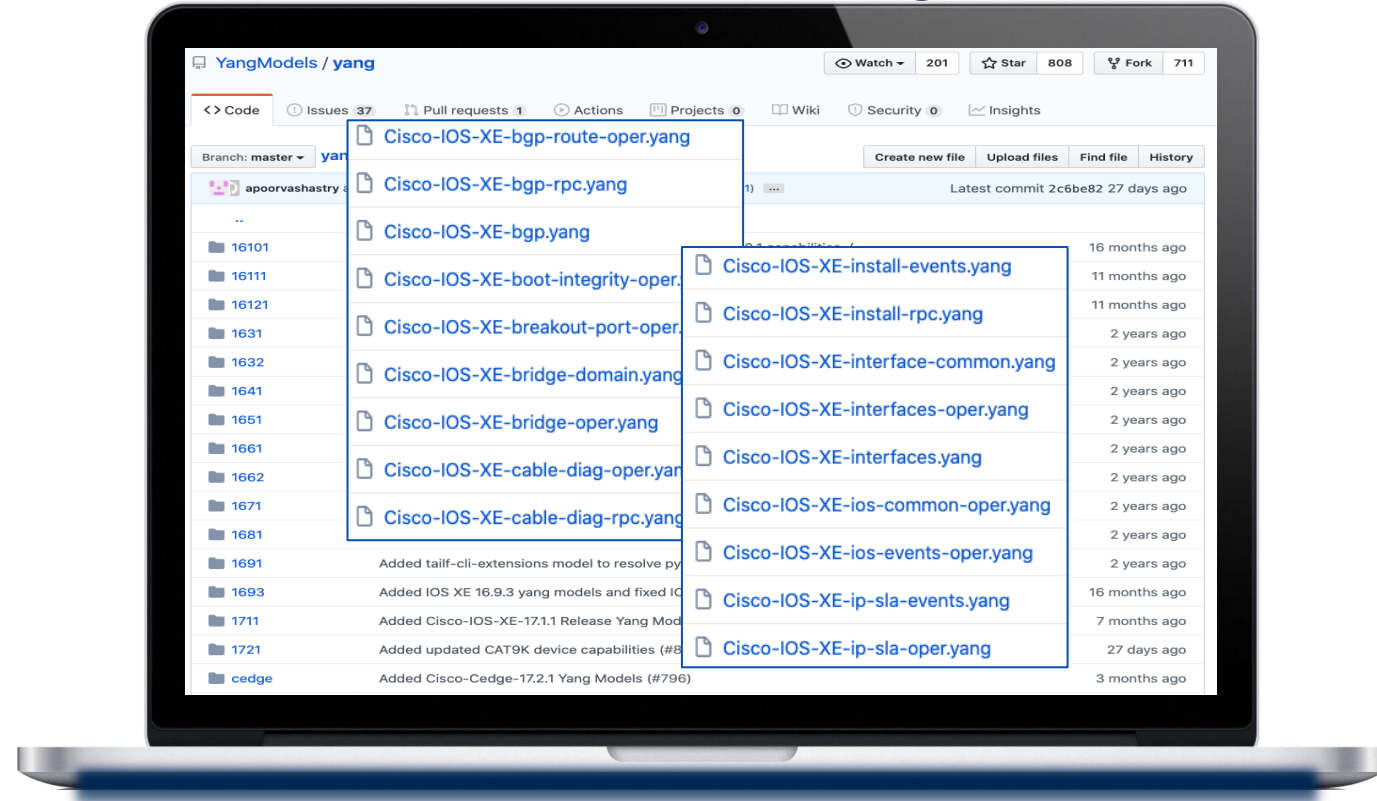
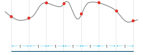


# IOS XE - YANG Model Coverage

Model Driven  
Programmability

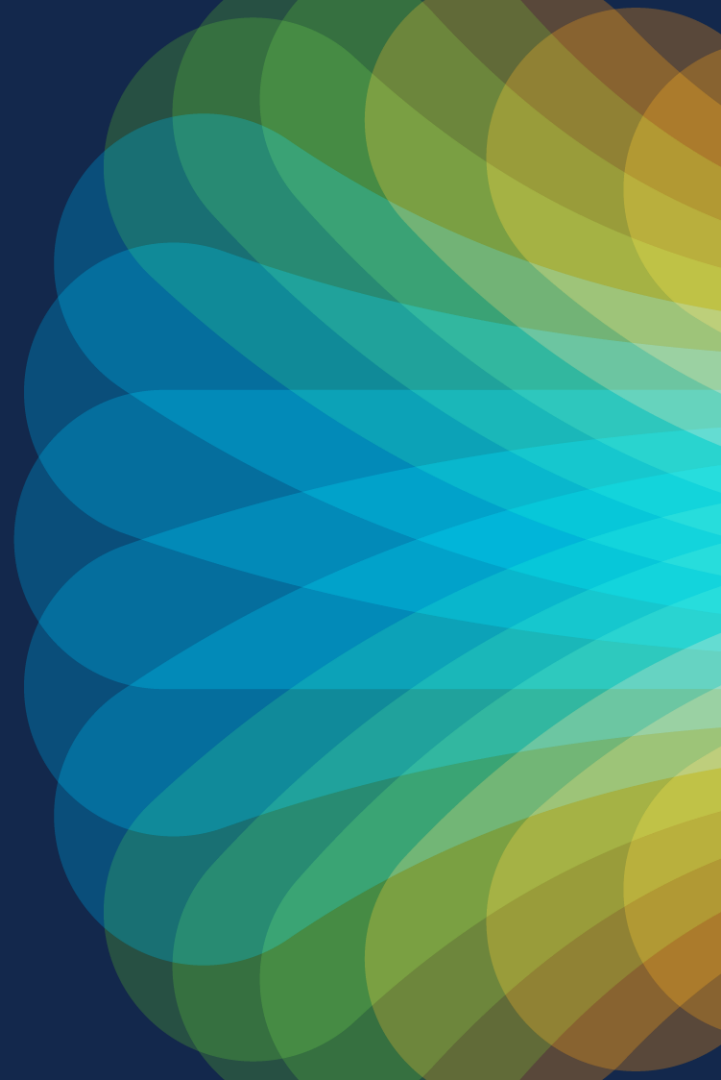


Model Driven  
Telemetry



<https://github.com/YangModels/yang/tree/master/vendor/cisco/xe>

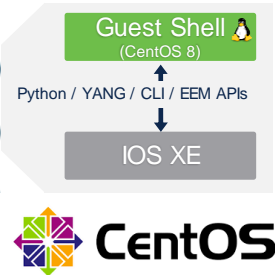
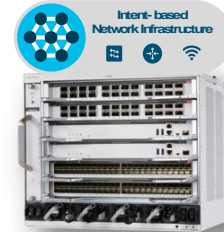
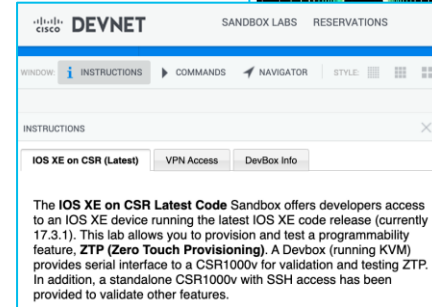
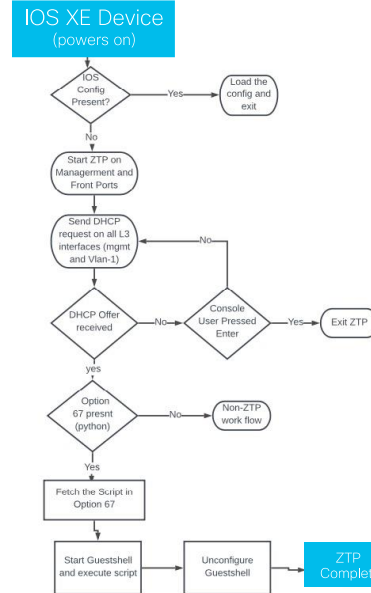
# Day 0



# Day 0 - Classic ZTP Overview

The Day 0 ZTP features are used to automatically configure and provision network devices.

1. When an IOS XE device boots and no configuration is present, the device will issue a DHCP request on the management port and on the front panel port.
2. If the DHCP response contains option 67 then ZTP is initiated and the device will retrieve and execute the python script from within the Guest Shell
3. Guest Shell is started and networking is



<https://www.youtube.com/watch?v=EAXnftG6odg>

<https://blogs.cisco.com/developer/device-provisioning-with-ios-xe-zero-touch-provisioning>

<https://devnetsandbox.cisco.com/RM/Diagram/Index/f2e2c0ad-844f-4a73-8085-00b5b28347a1?diagramType=Topology>

Secure ZTP blog <https://blogs.cisco.com/developer/secureztp01>


# ZTP Overview

1. When an IOS XE device boots and no configuration is present, the device will issue a DHCP request on the management port and on the front panel port.
2. If the DHCP response contains option 67 then ZTP is initiated and the device will retrieve and execute the python script from within the Guest Shell
3. Guest Shell is started and networking is automatically configured


<https://www.youtube.com/watch?v=EAXnftG6odg>

<https://blogs.cisco.com/developer/device-provisioning-with-ios-xe-zero-touch-provisioning>

Cisco Blogs




Cisco Blog > Developer



Developer

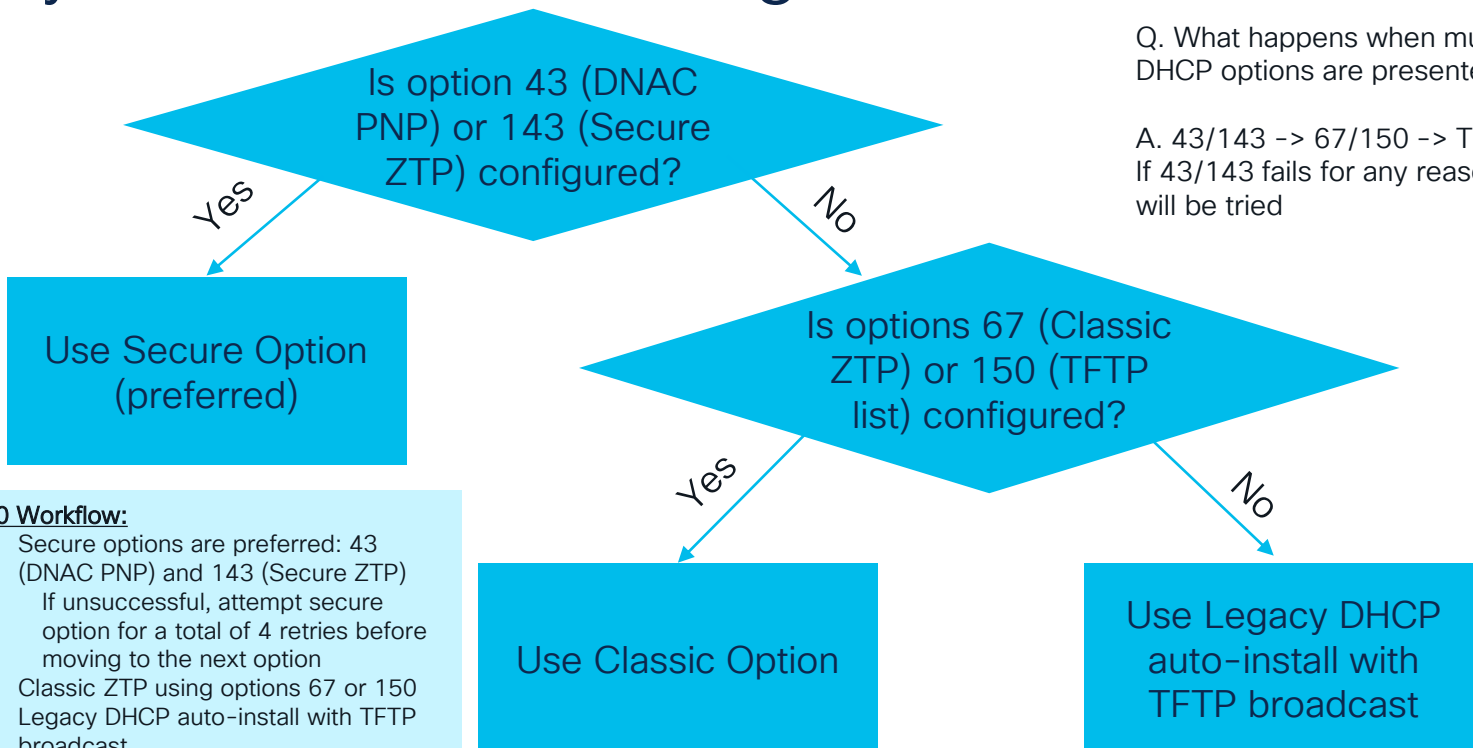
## Automate Device Provisioning with Cisco IOS XE Zero Touch Provisioning



Jeremy Cohoe  
April 15, 2019 - 1 Comment

When new hardware is ordered and it arrives on site, it's an exciting time. New hardware! New software! ... But new challenges too! But the age-old challenge of getting new devices on the network doesn't need to be one of them. Sitting in the lab pre-provisioning devices is no longer required if you're using Cisco IOS XE, because of features like Cisco Network Plug-n-Play (PnP) and Zero Touch Provisioning (ZTP). PnP is the premium solution made possible with Cisco DNA Center, while Zero Touch Provisioning (ZTP) is for the do-it-yourself customers who don't mind investing more time in configuring and maintaining the infrastructure required to bootstrap devices. IOS XE runs on the enterprise hardware and

# Day 0 device onboarding workflow



Q. What happens when multiple Day 0 DHCP options are presented to the device?

A. 43/143 -> 67/150 -> TFTP Broadcast  
If 43/143 fails for any reason, then 67/150 will be tried

## Day 0 Workflow:

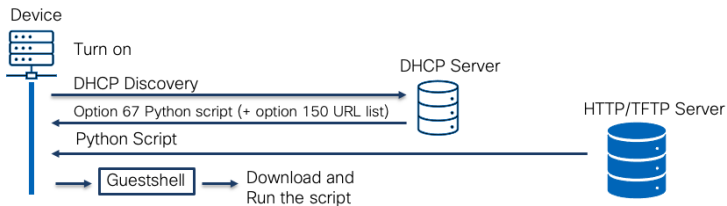
1. Secure options are preferred: 43 (DNAC PNP) and 143 (Secure ZTP)  
If unsuccessful, attempt secure option for a total of 4 retries before moving to the next option
2. Classic ZTP using options 67 or 150
3. Legacy DHCP auto-install with TFTP broadcast

# RFC8572 Secure ZTP

RFC details: <https://www.rfc-editor.org/rfc/rfc8572.html>

1. Conveyed Information: used to encode the redirect information and onboarding information (switch config)
2. Ownership Certificate: used by a device to verify the signature over the conveyed information
3. Ownership Voucher: used to verify a device owner as defined by the manufacturer (from the MASA)

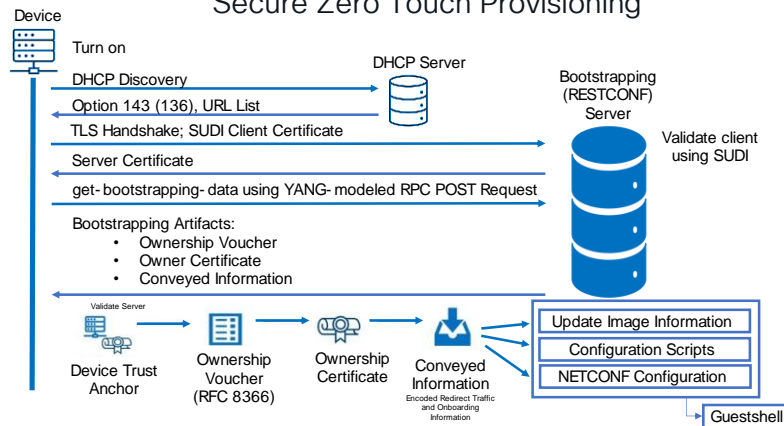
## Classic Zero Touch Provisioning



Some security requirements for classic ZTP are resolved using Secure ZTP:

- Management system needs to validate the device
- Device needs to validate the server
- Device must validate the data is what server sent

## Secure Zero Touch Provisioning



As part of the SZTP RFC, the device supports image upgrade as part of the conveyed information

# Classic vs Secure ZTP Example Output

## Classic ZTP

Acquired IPv4 address 10.1.1.157 on Interface Vlan1

Received following DHCPv4 options:

```
domain-name      : ios-xe-programmability-lab.cisco.com
bootfile         : http://10.1.1.3/ztp-simple.py
dns-server-ip    : 128.107.212.175
si-addr          : 10.1.1.3
```

## Secure ZTP

Acquired IPv4 address 192.168.1.61 on Interface  
GigabitEthernet0/0

Received following DHCPv4 options:

```
bootstrap-server-list : https://192.168.1.10:11601
```

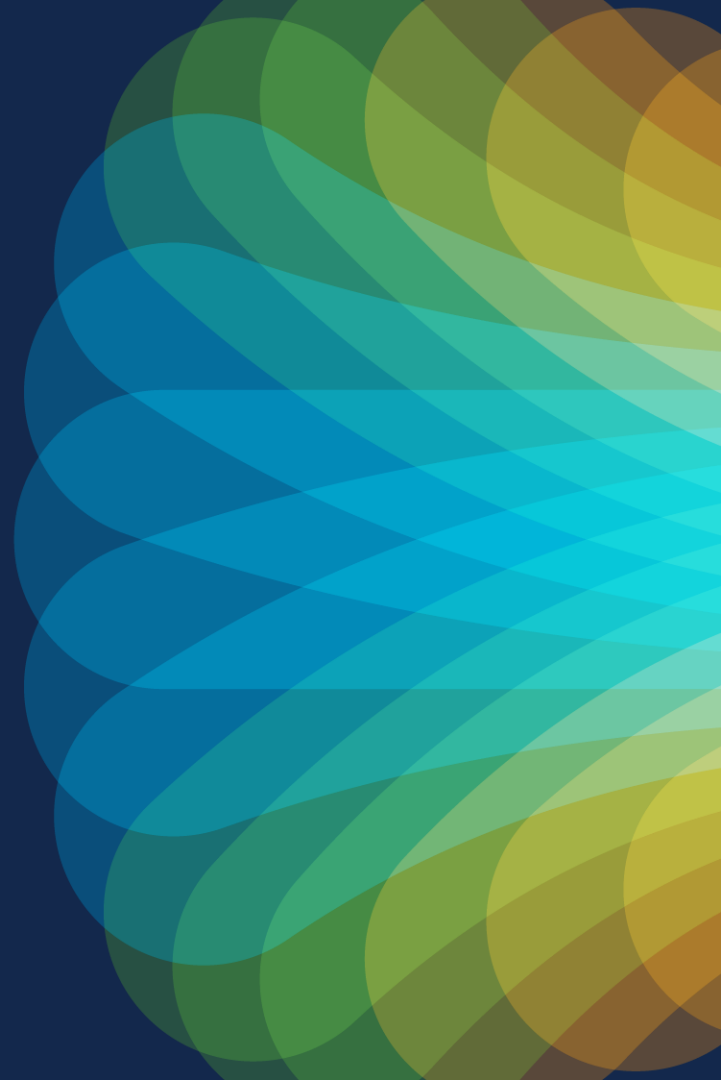
See additional info here: <https://github.com/jeremycohoe/IOSXE-Zero-Touch-Provisioning/blob/master/logs/pod08-xelab-vlan1.txt>

# Use Case: Image Upgrade from Conveyed Info

```
{
  "ietf-sztp-conveyed-info:onboarding-information" : {
    "boot-image" : {
      "os-name" : "VendorOS",
      "os-version" : "17.2R1.6",
      "download-uri" : [ https://example.com/path/to/image/file ],
      "image-verification" : [
        {
          "hash-algorithm" : "ietf-sztp-conveyed-info:sha-256",
          "hash-value" : "ba:ec:cf:a5:67:82:b4:10:77:c6:67:a6:22:ab:\
7d:50:04:a7:8b:8f:0e:db:02:8b:f4:75:55:fb:c1:13:b2:33"
        }
      ]
    },
    "configuration-handling" : "merge",
    "pre-configuration-script" : "base64encodedvalue==",
    "configuration" : "base64encodedvalue==",
    "post-configuration-script" : "base64encodedvalue=="
  }
}
```



# Day 1



# Build NETCONF XML Payload to SET Hostname

The screenshot displays the Cisco YANG Suite interface, which is used for managing network configurations. The interface is divided into several sections:

- Left Sidebar:** Contains navigation links for Admin, Setup, Analytics, Explore, Protocols, Test Manager, and Help.
- Top Bar:** Shows the current context as "YANG Suite / NETCONF / YANG set 'c9300-default-yangset' / Modules". It also includes a user profile icon labeled "admin" and a help icon.
- Main Content Area:**
  - YANG Set:** A dropdown menu showing "c9300-default-yangset".
  - Module(s):** A dropdown menu showing "Cisco-IOS-XE-native".
  - Load Module(s):** A button to load the selected module.
  - NETCONF Operation:** A dropdown menu showing "edit-config".
  - Device:** A dropdown menu showing "C9300".
  - Edit Device:** A button to edit the device configuration.
  - Open Device Window:** A dropdown menu.
  - Buttons:** A row of buttons including "YANG Tree", "Replays", "RPC Options...", "Build RPC", "Run RPC(s)", and "Clear RPC(s)".
  - Nodes:** A tree view showing the hierarchy of the "Cisco-IOS-XE-native" module. The "hostname" node is selected, and its value "netconf-pro" is displayed in a text box.
  - XML Payload:** A text area showing the generated NETCONF XML payload:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <hostname>netconf-pro</hostname>
      </native>
    </config>
  </edit-config>
</rpc>
```

# Demo: Generate Python from NETCONF

Access YANG Suite >  
Protocols > NETCONF

Select YANG model then  
Build RPC payload

Select Replays

Select Generate Python  
script

The screenshot displays the Cisco YANG Suite interface. On the left, a sidebar shows the navigation menu with 'Admin', 'Setup', 'Explore', 'Protocols', and 'Help'. The 'Protocols' section is expanded, showing 'NETCONF'. The main area is titled 'YANG Suite / NETCONF / YANG set "c9300-default-yangset" / Modules'. It shows the 'c9300-default-yangset' YANG set selected, with the 'Module(s)' dropdown set to 'Cisco-IOS-XE-native'. The 'NETCONF Operation' dropdown is set to 'edit-config', and the 'Device' dropdown is set to 'C9300'. The 'YANG Tree' view on the left shows a hierarchy of nodes, with 'ios-vlan:vlan-list' selected. The 'Replays' tab is active, showing a list of replays. A blue box highlights the 'Generate Python script' option in the 'Replays' menu. The 'RPC Options' tab is also visible. The 'Build RPC' button is highlighted. The 'Run RPC(s)' button is visible. The 'Clear RPC(s)' button is visible. The 'RPC' tab is active, showing the generated Python script. The script is a Python program that uses the 'argparse' module to parse command-line arguments and the 'ncclient' module to send an RPC request to a NETCONF server. The script includes a 'payload' dictionary containing the RPC request data. The script is as follows:

```
1 #!/usr/bin/env python
2 import lxml.etree as et
3 from argparse import ArgumentParser
4 from ncclient import manager
5 from ncclient.operations import RPCError
6
7 payload = [
8     ...
9     <get xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
10         <filter>
11             <lag-oper-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-lacp-o
12         </filter>
13     </get>
14     ...
15 ]
16
17 if __name__ == '__main__':
18
19     parser = ArgumentParser(description='Usage:')
20
21     # script arguments
22     parser.add_argument('-a', '--host', type=str, required=True,
23                         help="Device IP address or Hostname")
24     parser.add_argument('-u', '--username', type=str, required=True,
25                         help="Device Username (netconf agent username)")
26     parser.add_argument('-p', '--password', type=str, required=True,
27                         help="Device Password (netconf agent password)")
28     parser.add_argument('--port', type=int, default=830,
29                         help="Netconf agent port")
30     args = parser.parse_args()
```

# NETCONF + Ansible

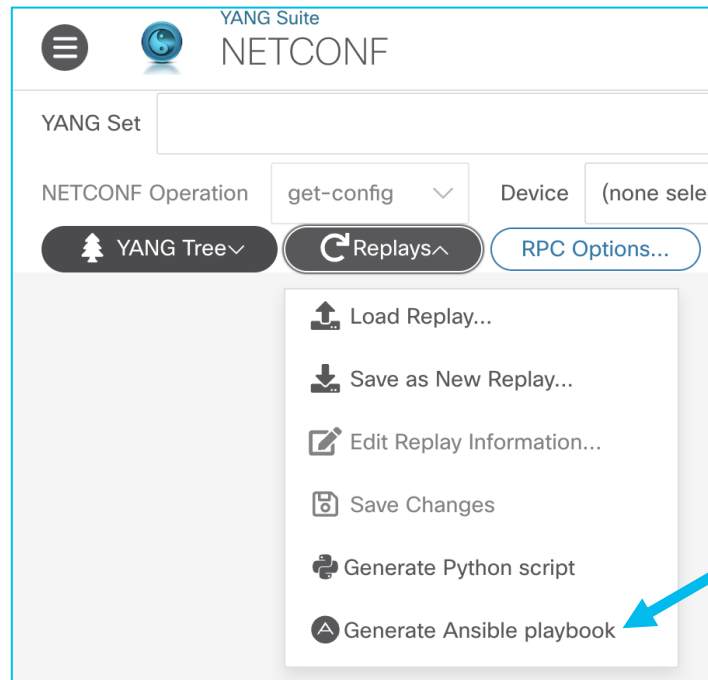
Similar to the “Generate Python” functionality, a new functionality generates YAML formatted for Ansible.

## Requirements

1. Install Ansible
2. Install NETCONF collection:  
`ansible-galaxy collection install NETCONF`

```
- name : conf-host
  hosts: c9300
  connection: netconf
  gather_facts: no

  tasks:
    - name: hostname-conf
      netconf_config:
        xml: |
          <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
            <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
              <hostname>c9300-pod29</hostname>
            </native>
          </config>
```



# NETCONF + Ansible Update Interface Description Demo

Cisco YANG Suite

Admin Setup Analytics Explore Protocols Test Manager Help

YANG Suite / Help  
Help: Welcome to Cisco YANG Suite!

admin

## Welcome to Cisco YANG Suite!

YANG Suite is a set of tools related to YANG models (RFC 6020, RFC 7950) and related technologies such as NETCONF (RFC 6241).

It provides a modular infrastructure which various YANG application plugins can be used.

Please check [developer.cisco.com/yangsuite](https://developer.cisco.com/yangsuite) for information, learning labs, and announcements.

## Docker-based Installation

The `yangsuite/docker/start_yang_suite.sh` script performs the following:

- Prompts for username, password, and email which will be the superuser to yangsuite.
- Gives the choice of creating test SSL/TLS certificate and key.
- Creates an environment file needed for the yangsuite docker container install.
- Runs `docker-compose up`.

## YANG Suite Documentation

Search docs

- Welcome to Cisco YANG Suite!
- Device Profiles
  - Managing device profiles
  - Setting up YANG Suite TLS/SSL
- File Manager
  - Constructing and populating a YANG module repository
  - Uploading YANG files from the local filesystem to a YANG repository
  - Downloading YANG files via NETCONF from a device to a YANG repository
  - Copying YANG modules via SCP from a server to a YANG repository
  - Importing YANG modules from a Git repository to a YANG repository
  - Defining a YANG module set
  - Managing YANG module files in YANG Suite
- Working with YANG Models
  - Exploring YANG Models
- YANG Suite Analytics
  - Datasets
  - Mapping SNMP OIDs to YANG XPath
- Using gNMI with YANG Suite
- YANG Suite gRPC telemetry receivers
  - Telemetry over gRPC Clear Channel
  - Configuring gRPC Telemetry Receivers
  - Telemetry over gRPC Secure Channel
- Using NETCONF with YANG Suite
  - Using NETCONF RPCs
  - Working with NETCONF Notification Streams
  - Using Replays for repeated workflows
  - Locking and unlocking datastores
- RESTCONF in YANG Suite
  - YANG Suite RESTCONF
- Test management with YANG Suite
  - Using Test Manager to define and execute tests
  - Generating YANG model Tests
  - Managing and editing replay files

# YANG Suite RESTCONF Demo

Cisco YANG Suite

Admin

Setup

Analytics

Explore

Protocols

Test Manager

Help

YANG Suite / Help

Help: Welcome to Cisco YANG Suite!

admin

Welcome to Cisco YANG Suite!

YANG Suite is a set of tools related to YANG models ([RFC 6020](#), [RFC 7950](#)) and related technologies such as NETCONF ([RFC 6241](#)).

It provides a modular infrastructure which various YANG application plugins can be used.

Please check [developer.cisco.com/yangsuite](#) for information, learning labs, and announcements.

## Docker-based Installation

The `yangsuite/docker/start_yang_suite.sh` script performs the following:

- Prompts for username, password, and email which will be the superuser to yangsuite.
- Gives the choice of creating test SSL/TLS certificate and key.
- Creates an environment file needed for the yangsuite docker container install.
- Runs `docker-compose up`.

YANG Suite Documentation

Search docs

- Welcome to Cisco YANG Suite!
- Device Profiles
  - Managing device profiles
  - Setting up YANG Suite TLS/SSL
- File Manager
  - Constructing and populating a YANG module repository
  - Uploading YANG files from the local filesystem to a YANG repository
  - Downloading YANG files via NETCONF from a device to a YANG repository
  - Copying YANG modules via SCP from a server to a YANG repository
  - Importing YANG modules from a Git repository to a YANG repository
  - Defining a YANG module set
  - Managing YANG module files in YANG Suite
- Working with YANG Models
  - Exploring YANG Models
- YANG Suite Analytics
  - Datasets
- Using gNMI with YANG Suite
- YANG Suite gRPC telemetry receivers
  - Telemetry over gRPC Clear Channel
  - Configuring gRPC Telemetry Receivers
  - Telemetry over gRPC Secure Channel
- Using NETCONF with YANG Suite
  - Using NETCONF RPCs
  - Working with NETCONF Notification Streams
  - Using Replays for repeated workflows
  - Locking and unlocking datastores
- RESTCONF in YANG Suite
  - YANG Suite RESTCONF
- Test management with YANG Suite
  - Using Test Manager to define and execute tests
  - Convert Ytool Test Suites to YANG Suite Formats to activate Windows.
  - Generating YANG model Tests
  - Importing and exporting YANG Suite tests

# CLI to YANG

This new CLI addition to “show run | format” brings additional visibility into the YANG modelled configuration, either for NETCONF with XML or JSON with RESTCONF  
Easily convert CLI into YANG to re-use in tooling, scripts, and automation and orchestration systems

```
show run | format netconf-xml
show run | format restconf-json
```

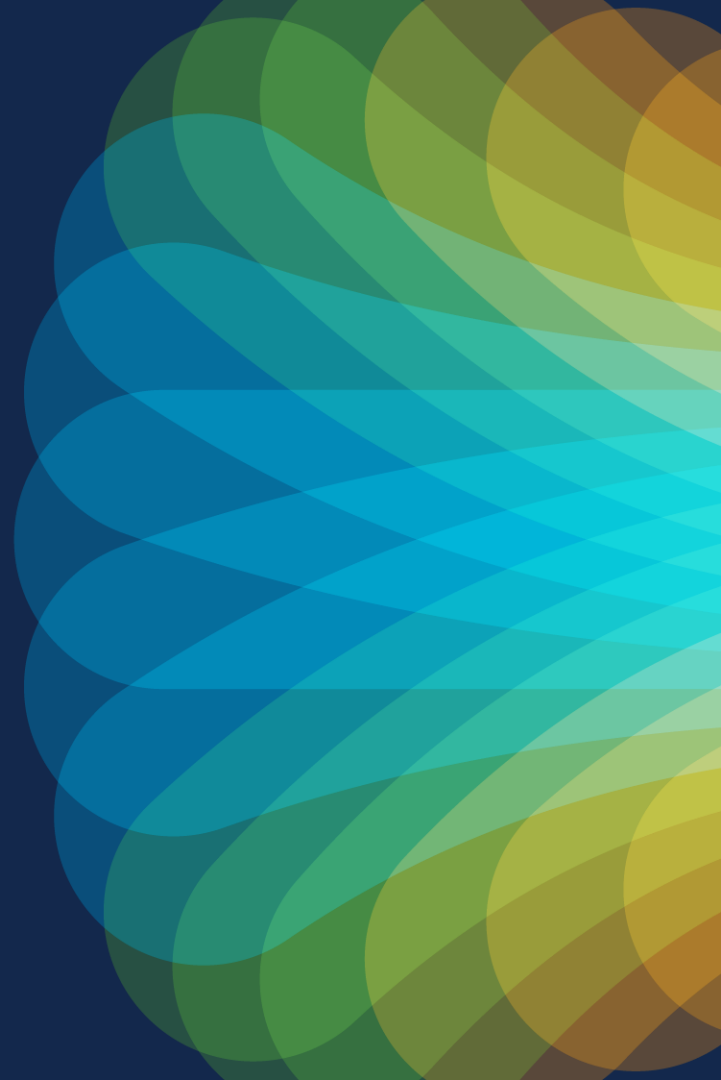
```
C9300#show run | format netconf-xml
<config xmlns="http://tail-f.com/ns/config/1.0">
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <version>17.7</version>
    <memory>
      <free>
        <low-watermark>
          <processor>131752</processor>
        </low-watermark>
      </free>
    </memory>
  </native>
</config>
```

```
C9300#show run | format restconf-json
{
  "data": {
    "Cisco-IOS-XE-native:native": {
      "version": "17.7",
      "memory": {
        "free": {
          "low-watermark": {
            "processor": 131923
          }
        }
      }
    }
  }
}
```

```
C9300#
C9300#show run | i netconf-yang
netconf-yang
C9300#
```

Requires `netconf-yang` Data Model Interfaces to be enabled  
CLIs with corresponding native YANG and modeled in show run are returned

# Day 2

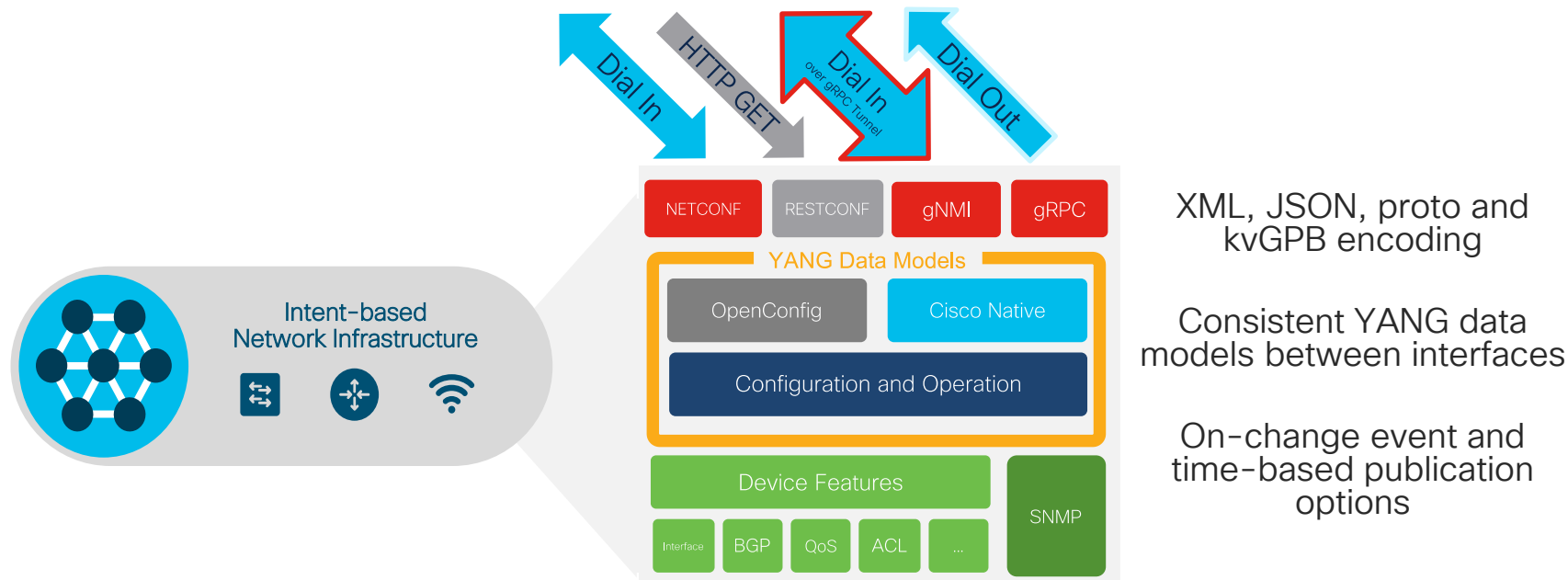




# Model Driven Telemetry Interfaces

- ↔ 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



# IOS XE Model Driven Telemetry

Cisco IOS XE



CLI

...or with...

YANG

gNMI Dial-In/Dynamic NETCONF Dial-In ↔ gRPC Dial-Out/Configured



Collector/Receiver  
Decodes to text



Storage  
Time Series Database



Monitoring  
and Visualizations



[https://hub.docker.com/r/jeremycohoe/tig\\_mdt](https://hub.docker.com/r/jeremycohoe/tig_mdt)

<https://github.com/jeremycohoe/cisco-ios-xe-mdt>

[https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/179/b\\_179\\_programmability\\_cg/m\\_179\\_prog\\_ietf\\_telemetry.html](https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/179/b_179_programmability_cg/m_179_prog_ietf_telemetry.html)



# Updated TIG\_MDT container now available!

Updated April 2023

Upgrade coming to Telegraf, Influx, and Grafana Model Driven Telemetry (TIG\_MDT) Docker container

Making it easier to consume telemetry in production

Upgraded Telegraf, InfluxDB, and Grafana tools

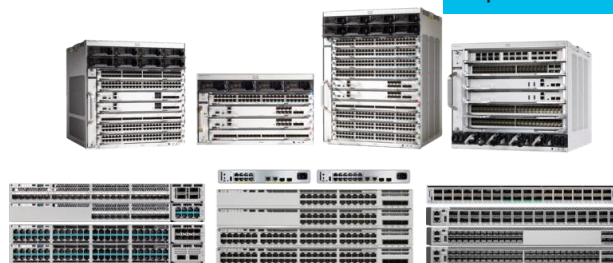
Additional dashboards for

Device Health, Wireless Client, Wireless AP, RF etc

Examples for device CLI configuration for telemetry

Details of scale and data storage requirements

```
docker pull jeremycohoe/tig_mdt
docker run -ti -p 3000:3000 -p 57500:57500 jeremycohoe/tig_mdt
```



Collector/Receiver  
Decodes to text



Storage  
Time Series Database



Monitoring  
and Visualizations



[https://hub.docker.com/r/jeremycohoe/tig\\_mdt](https://hub.docker.com/r/jeremycohoe/tig_mdt)

<https://github.com/jeremycohoe/cisco-ios-xe-mdt>

[https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/179/b\\_179\\_programmability\\_cg/m\\_179\\_prog\\_ielf\\_telemetry.html](https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/179/b_179_programmability_cg/m_179_prog_ielf_telemetry.html)

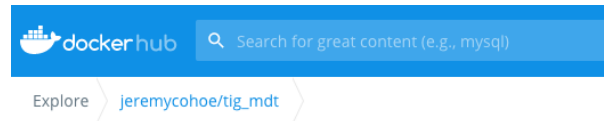


# Telemetry Quick Start with Docker

1. Download and start the Docker container from [https://hub.docker.com/r/jeremycohoe/tig\\_mdt](https://hub.docker.com/r/jeremycohoe/tig_mdt)



```
$ docker pull jeremycohoe/tig_mdt
$ docker run -dit -p 3000:3000 -p 57500:57500 tig_mdt /start.sh
```



2. Configure the IOS XE device to send telemetry <https://github.com/jeremycohoe/cisco-ios-xe-mdt>

```
telemetry ietf subscription 102
encoding encode-kvgpb
filter xpath /interfaces-ios-xe-oper:interfaces/interface
source-address 10.85.134.65
stream yang-push
update-policy periodic 2000
receiver ip address 10.85.134.66 57500 protocol grpc-tcp
```

3. Access the Grafana interface on port 3000



# Cisco Telemetry Data Broker (Telegraf)

Cisco Telemetry Broker provides many benefits include brokering, filtering, and transforming data. It provides the ability to replicate telemetry data.

- Cisco Secure Network Analytics (Stealthwatch) UDP Director (UDPD) replicates UDP traffic to multiple destinations.
- Cisco Telemetry Broker
  - Builds upon UDPD
  - Optimizes telemetry pipelines for the hybrid cloud
  - Simplifies the consumption of telemetry data for customers' business-critical tools by brokering hybrid cloud data, filtering unneeded data, and transforming data to a usable format



## ► Brokering Data:

The ability to route and replicate telemetry data from a source location to multiple destination consumers.

Quickly onboard new telemetry-based tools!

## ► Filtering Data:

The ability to filter data that is being replicated to consumers for fine grain control over what consumers are able to see and analyze.

Save money sending data to expensive tools!

## ► Transforming Data:

The ability to transform data protocols from the exporter to the consumer's protocol of choice.

Enable tools to consume multiple data formats!

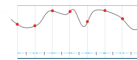
PRODUCTS	UNIT LIST PRICE	STATUS	QUANTITY	ACTION
Virtual License Cisco Telemetry Broker Essential License 100GB/Day 10 000 000 000	\$10.00 Per 100GB/Day/12 Months	Active	20 100GB	Revoke
Support - Virtual License Enhanced Support for Cisco Telemetry Broker SWS-19-SUP-E	200.00 Per Each/Month	Active	1 Each	Revoke
Premium Support for Cisco Telemetry Broker SWS-19-SUP-P	Take an action to see the List Price Per Each/Month	Not Active	1 Each	Revoke

<https://cs.co/telemetrybroker> aka <https://www.cisco.com/c/en/us/products/security/telemetry-broker/index.html>  
<https://blogs.cisco.com/security/taking-full-control-of-your-telemetry-with-the-intelligent-telemetry-plane>

# Model Driven Telemetry Interface Comparison

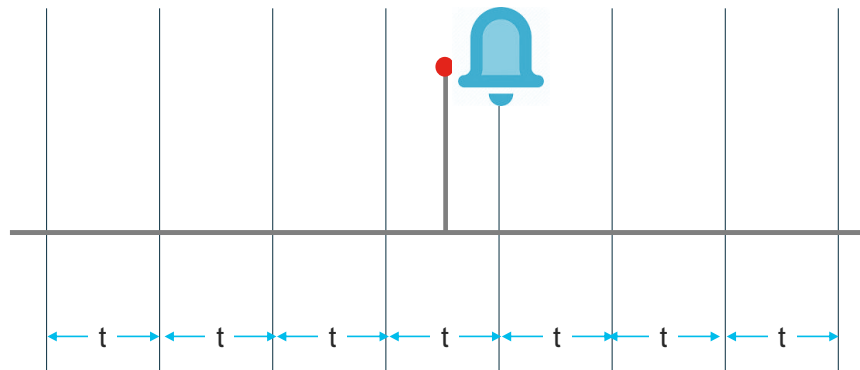
	NETCONF	gRPC (Dial-Out)	gNMI
Minimum IOS XE Version	16.6	16.10	Dial-In: 16.12 over gRPC tunnel: 17.11
Recommended Version	17.9	17.9	Dial-In: 17.9 over gRPC tunnel 17.11
Telemetry Direction	Dial-In, IOS XE is server	Dial-Out IOS XE is client	Dial-In IOS XE is server Dial-Out gRPC Tunnel
Configuration	Dynamic per session	Static per configuration	Dynamic per session
Telemetry Collector	Client	Server	Client
Encoding	XML	KV GPB	JSON_IETF + PROTO
Security	SSH + PKI certificate or password	mTLS or plain-text	mTLS certificates mTLS cert only or mTLS cert + user/pass authentication
Transport Protocol	SSH	HTTP2	HTTP2
Data Models	YANG	YANG	YANG

Network architecture, security posture and policy, YANG data modules, tools and language preferences, and standards, and software version, are some considerations when leveraging the various MDT interfaces



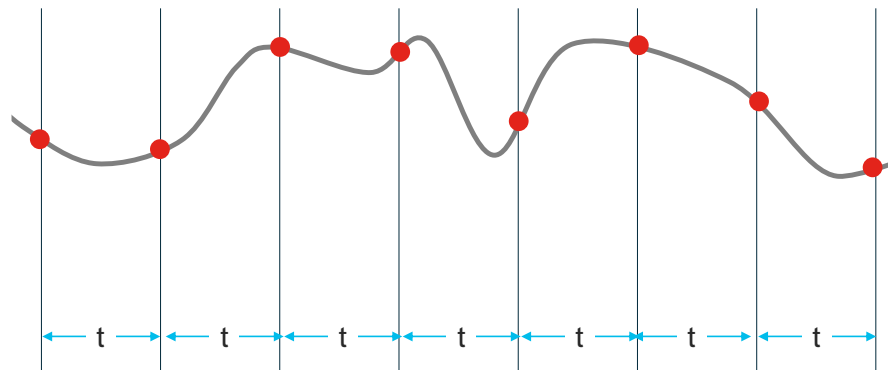
# Publication options

## On-Change



Feature Model "On-Change" Notifications  
Event Notifications (failed login, optic fault, etc)  
State and Configuration

## Periodic



Feature Model "Periodic" Notifications  
Time based publication  
Minimum interval 100 centiseconds (1s)

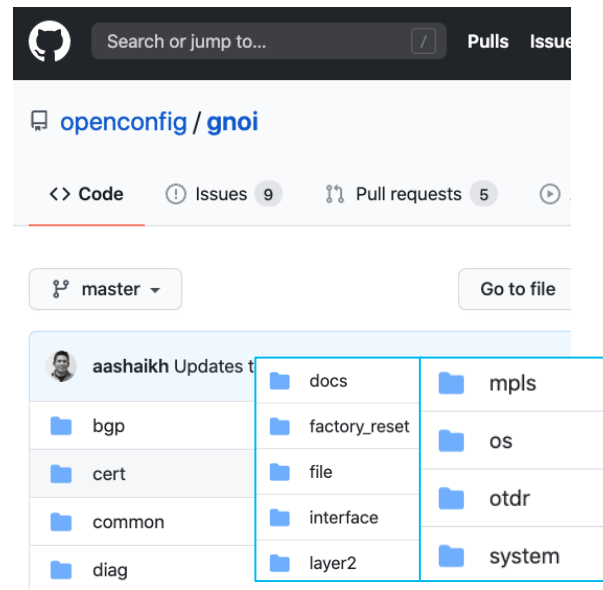
OpenConfig YANG with GNXI, not NETCONF

# gNOI – gNMI / 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 wipe	write erase ...	17.7
File.proto	Not implemented	copy, delete	N/A
System.proto	Not implemented	reload, set boot	N/A

<https://github.com/openconfig/gnoi>





# Migrating from SNMP to gRPC Dial-Out Telemetry

What is the expected increase in CPU/Memory when using the gRPC Dial-Out telemetry interface, compared to SNMP ?

gRPC adds 2% for each telemetry collector  
SNMP adds 6% and an additional 4% for each collector

Testbed:

Ubuntu Linux VM  
Telegraf Tooling  
SNMP + gRPC

C9300-48

Spirent 48 port  
traffic generator

## CPU impact with multiple gRPC receivers

1. No Interface collection, CPU @ 1%
2. Add gRPC 1 receiver, CPU @ 3% +2
3. Add 2<sup>nd</sup> gRPC receiver, CPU @ 5% +2
4. Add 3<sup>rd</sup> gRPC receiver, CPU @ 8% +3

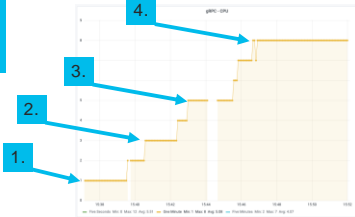
### gRPC configuration

```
telemetry intf subscription 1000
encoding json-ietf
filter xpath /interface-cs=sw-oper:interface/interface
source-address 128.107.223.250
stream yang-push
update-policy periodic 500
receiver ip address 10.10.10.10 v3500 protocol gRPC-top

telemetry intf subscription 1001
encoding json-ietf
filter xpath /interface-cs=sw-oper:interface/interface
source-address 128.107.223.250
stream yang-push
update-policy periodic 500
receiver ip address 10.10.10.10 v3500 protocol gRPC-top

telemetry intf subscription 1002
encoding json-ietf
filter xpath /interface-cs=sw-oper:interface/interface
source-address 128.107.223.250
stream yang-push
update-policy periodic 500
receiver ip address 10.10.10.10 v3500 protocol gRPC-top
```

© 2020 Cisco and/or its affiliates. All rights reserved. Cisco Confidential



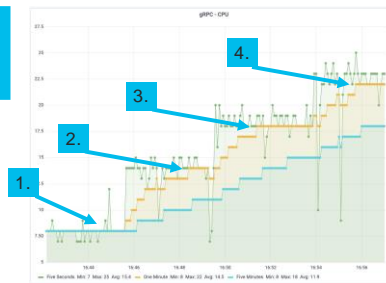
## CPU impact multiple SNMP pollers

```
telegraf process, set
$ snmpwalk -v 1 -c EN-TMR-Cisco123 128.107.223.250
```

1. Baseline CPU @ 8 %
2. Add 1 SNMP CPU @ 14% +6
3. Add 2<sup>nd</sup> SNMP CPU @ 18% +4
4. Add 3<sup>rd</sup> SNMP CPU @ 22% +4

### SNMP collection configuration

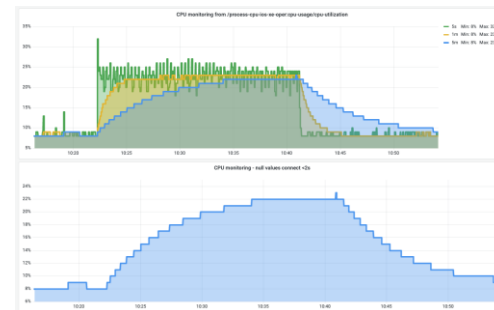
```
[[{"host": "128.107.223.250", "community": "public", "version": 1, "timeout": 10, "maxOids": 0, "maxRepetitions": 100, "bulk": true, "ignoreErrors": true, "log": true, "tags": {"host": "128.107.223.250", "community": "public", "version": 1, "timeout": 10, "maxOids": 0, "maxRepetitions": 100, "bulk": true, "ignoreErrors": true, "log": true}}]]
```



# Model Driven Telemetry: usage comparison

60-minute collection sample with 60-second update interval

Interface	CPU Impact	PCAP file size/data size (MB)	Data byte Rate	Data bit rate	Average Packet Rate (sec)	Average Packet Size (bytes)
gNMI	+3%	23 MB	6 kBps	53 kbps	5	1180
gRPC	+3%	69 MB	19 kBps	155 kbps	58	333
NETCONF	+2%	83 MB	23 kBps	185 kbps	29	780
RESTCONF	+4%	200 MB	35 kBps	281 kbps	37	945
SNMP *	+6%	120 / 87	24 kBps	197 kbps	90	273



17 xps collected at 60 second update interval

```

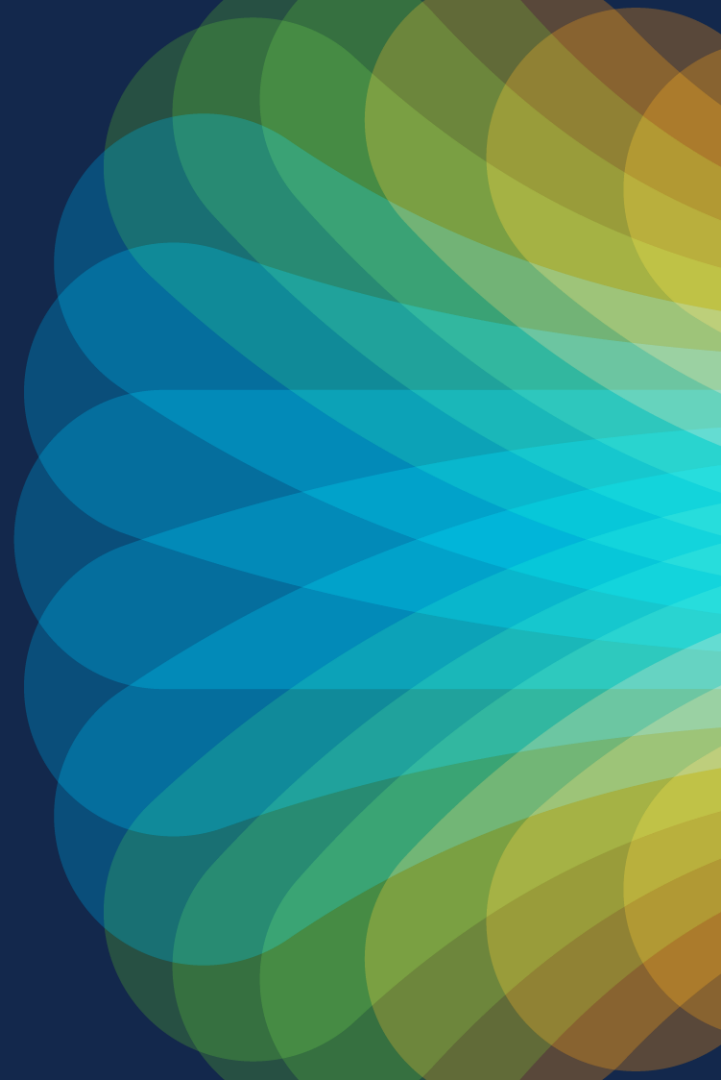
/arp-ios-xe-oper:arp-data
/cdp-ios-xe-oper:cdp-neighbor-details
/environment-ios-xe-oper:environment-sensors
/if:interfaces-state
/interfaces-ios-xe-oper:interfaces/interface
/ios:native
/ldp-ios-xe-oper:ldp-entries
/matm-ios-xe-oper:matm-oper-data
/mdt-oper:mdt-oper-data/mdt-subscriptions
/memory-ios-xe-oper:memory-statistics/memory-statistic
/oc-if:interfaces/interface/state/counters
/oc-platform:components
/oc-sys:system
/platform-ios-xe-oper:components
/poe-ios-xe-oper:poe-oper-data/poe-switch
/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization
/process-memory-ios-xe-oper:memory-usage-processes
  
```

+ Device-hardware-oper + Switch-stack-oper + more ?



This demonstrates that even when SNMP is only measuring Interfaces the load is still significantly higher than YANG which is measuring significantly more YANG data

# Day N



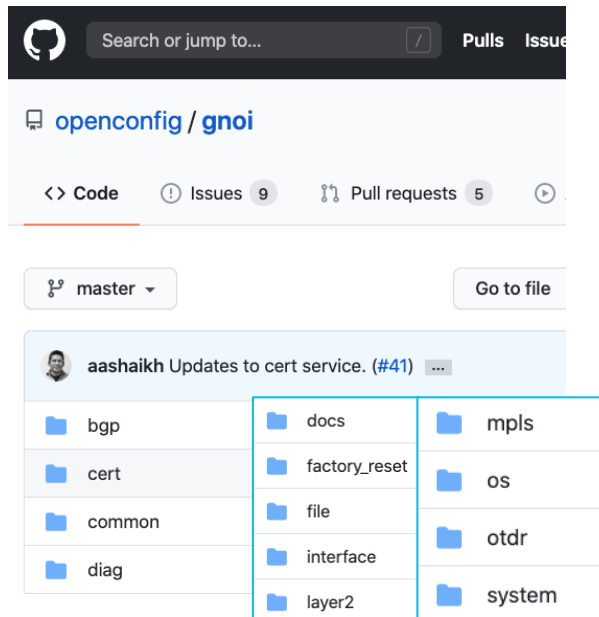
# gNOI

## 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

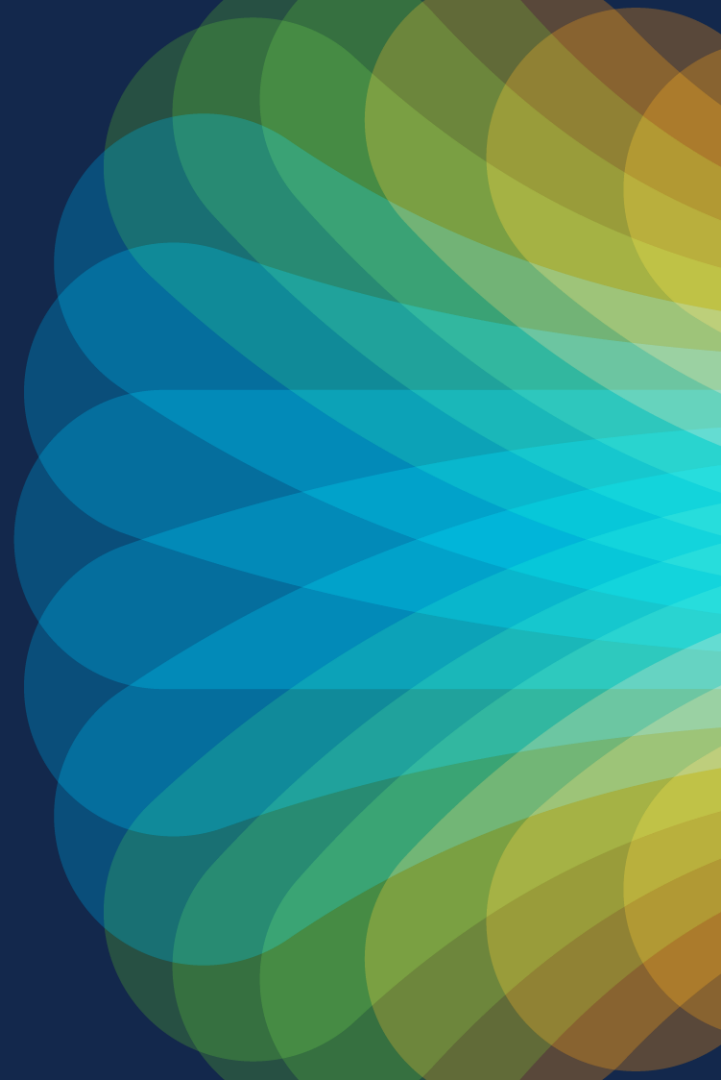
Protobuf RPC	Use	Related CLI	Release
Cert.proto	TLS Certificate management	crypto pki ...	17.3
Os.proto	Network Operating System management	install add file ...	17.5
Reset.proto	Factory Reset and wipe	write erase	17.7
File.proto	Not implemented	copy, delete	N/A
System.proto	Not implemented	reload, set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

<https://github.com/openconfig/gnoi>



gNOI cert.proto

# Certificate Management Service



# 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”



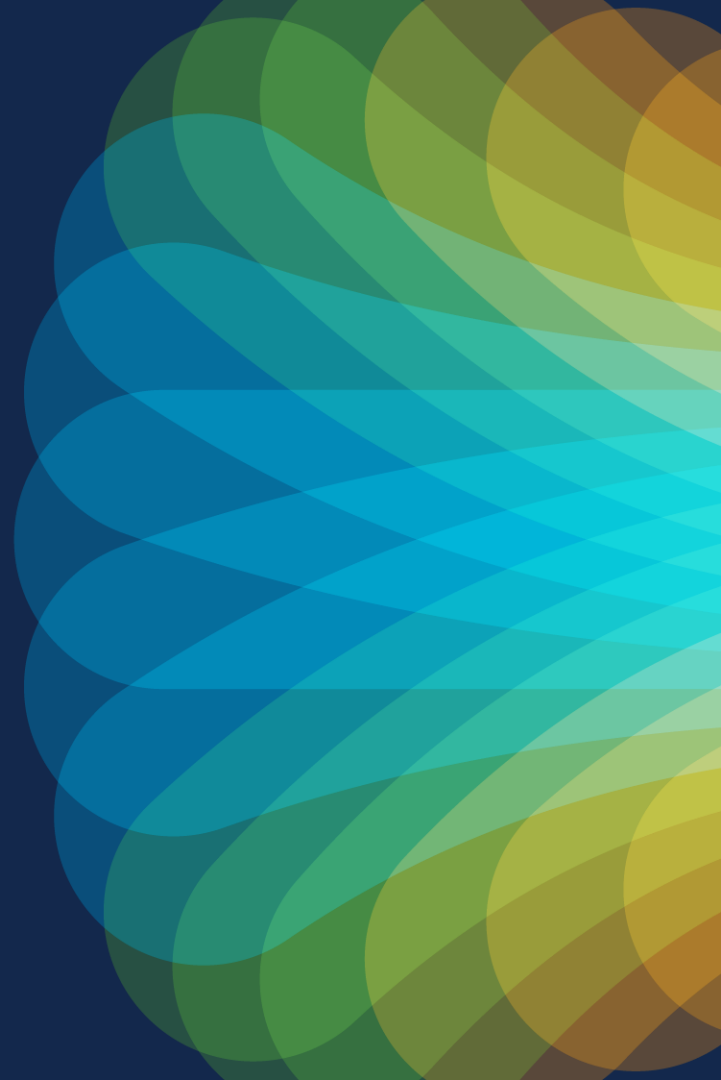
✕ auto@automation: ~ (ssh)

```
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 OS.proto Operating System API





# gNOI os.proto – Operating System API

OS installation, activation, and verification API

[https://github.com/google/gnxi/tree/master/gnoi\\_os](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

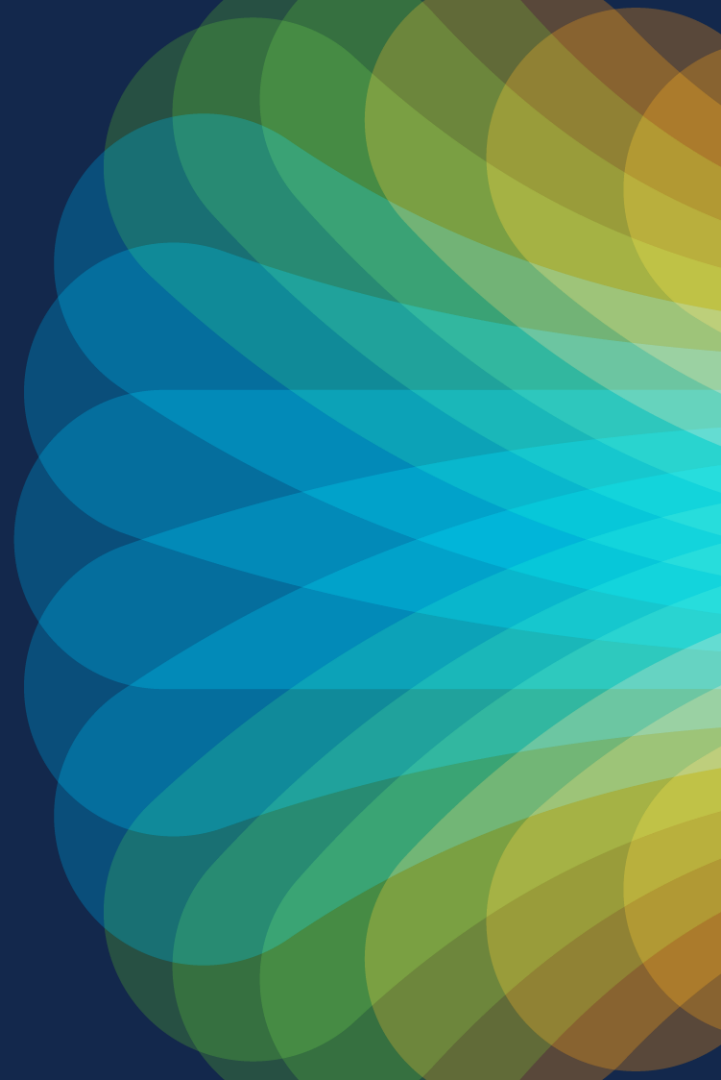
```
go get github.com/google/gnxi/gnoi_os
go install github.com/google/gnxi/gnoi_os
```

### 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

reset.proto 17.7



# gNOI reset.proto

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/](https://github.com/openconfig/gnoi/blob/master/factory_reset/) with tooling from [https://github.com/google/gnxi/tree/master/gnoi\\_reset](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 17.7.1

# gNOI reset.proto - tooling

[https://github.com/google/gnxi/tree/master/gnoi\\_reset](https://github.com/google/gnxi/tree/master/gnoi_reset)

Similar to previous gNOI implementations the recommended tooling is gnoi\_reset from Google's gNxl 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

```
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:~$ █
```

```

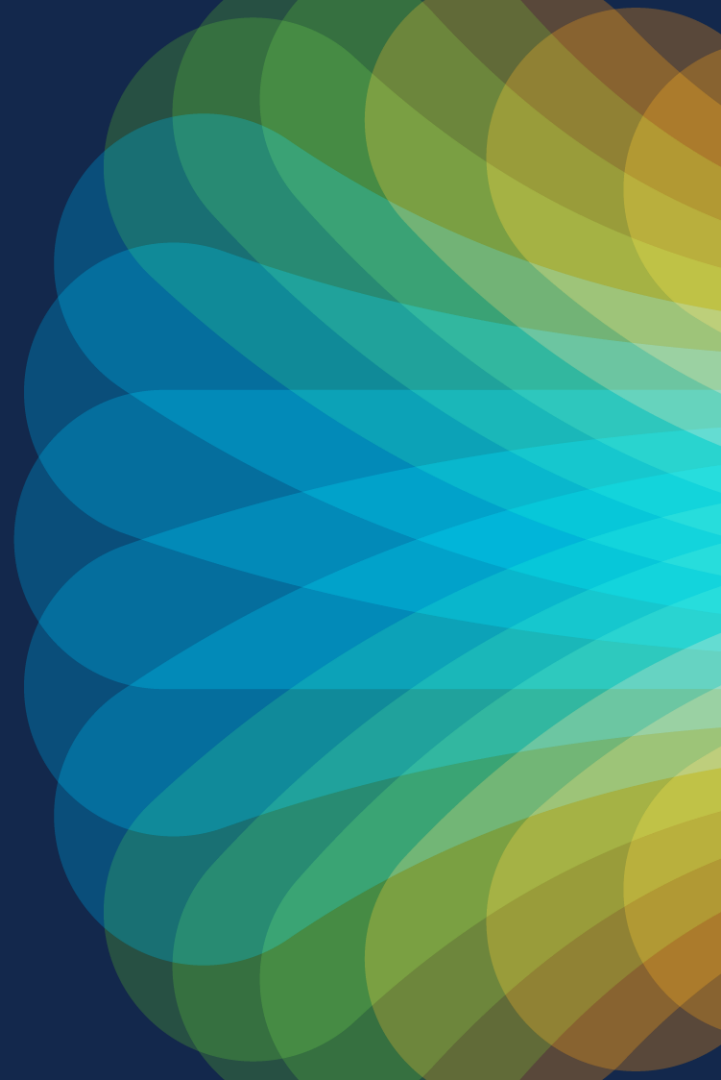
C9300#
Chassis 1 reloading, reason - Factory Reset
Oct 14 11:50:31.175: %PMAN-5-EXITACTION: F0/0: pvp: Process manager is exiting: reload fp action requested
Oct 14 11:50:32.839: %PMAN-5-EXITACTION: R0/0: pvp: Pger is exiting: rp processes exit with reload switch code

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]
```

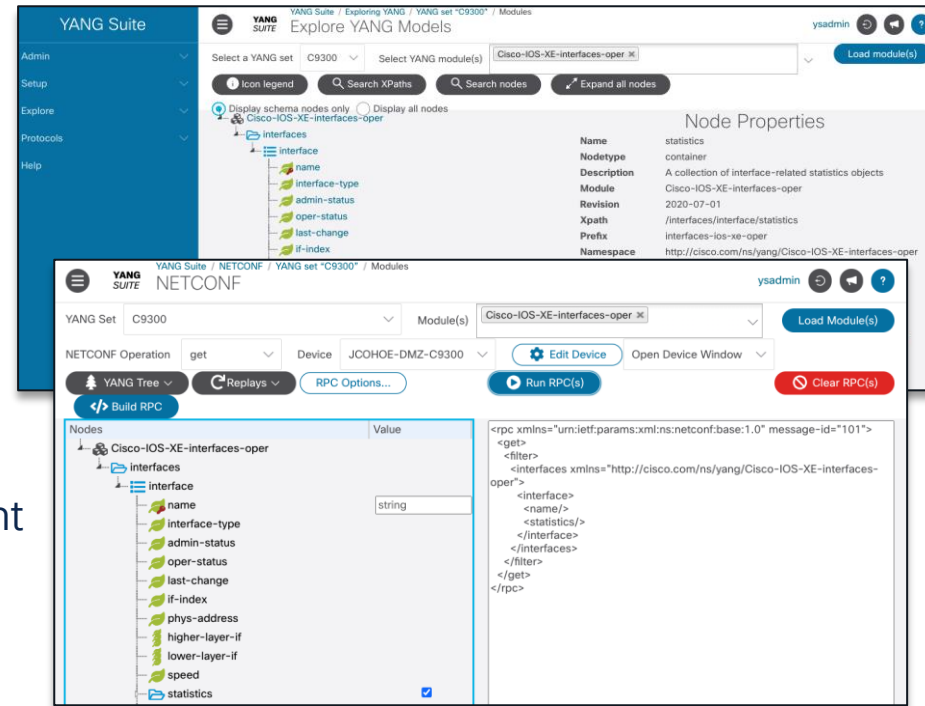
# Tooling YANG Suite Terraform



# 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



**Now Generally Available !**

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

[github.com/CiscoDevNet/yangsuite](https://github.com/CiscoDevNet/yangsuite)

# What's Included

- 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

## Core plugins

## Additional plugins

### Cisco YANG Suite

#### Admin

- Manage users
- Manage plugins
- View logs

#### Setup

- YANG files and repositories
- YANG module sets
- Device profiles

#### Analytics

- Datasets and diffs
- SNMP to YANG Mapping

#### Protocols

- gNMI
- gRPC telemetry
- NETCONF
- RESTCONF

#### Explore

- YANG



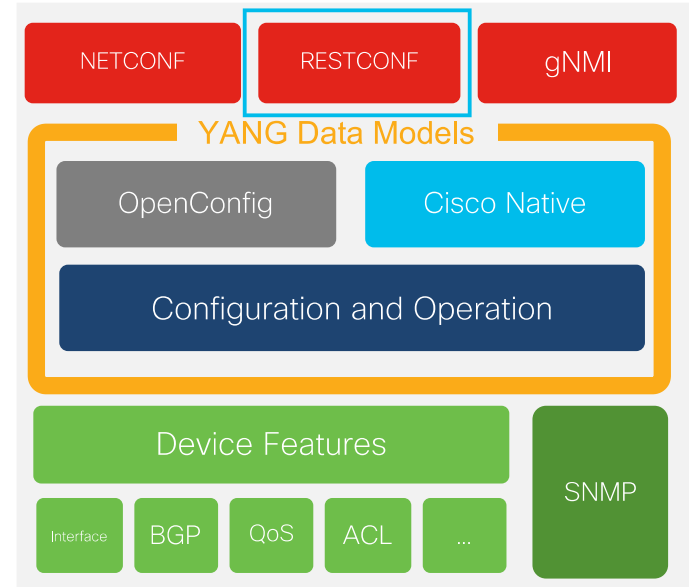
# Terraform is...



Infrastructure as Code (IaC) Software Tool providing a consistent CLI workflow to manage hundreds of cloud services. Terraform codifies cloud APIs into declarative configuration files.

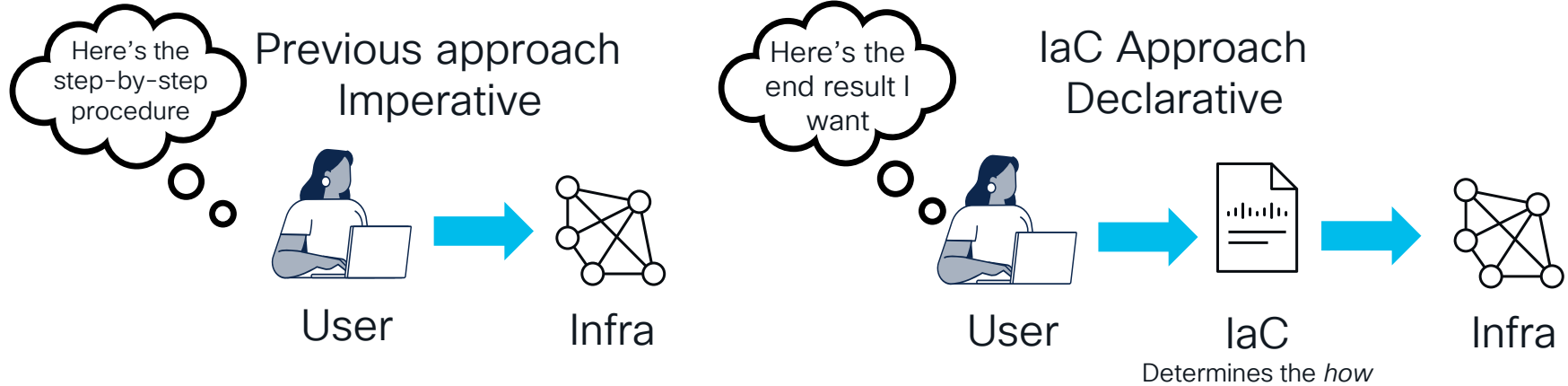
- Cloud Native Tooling circa 2014 from HashiCorp
- Agentless, single binary file
- Zero server-side dependencies

## Terraform uses the RESTCONF API



# What's IaC?

Infrastructure as Code (IaC) is the process of managing changes through code, rather than a manual process



Learn more about IaC here:

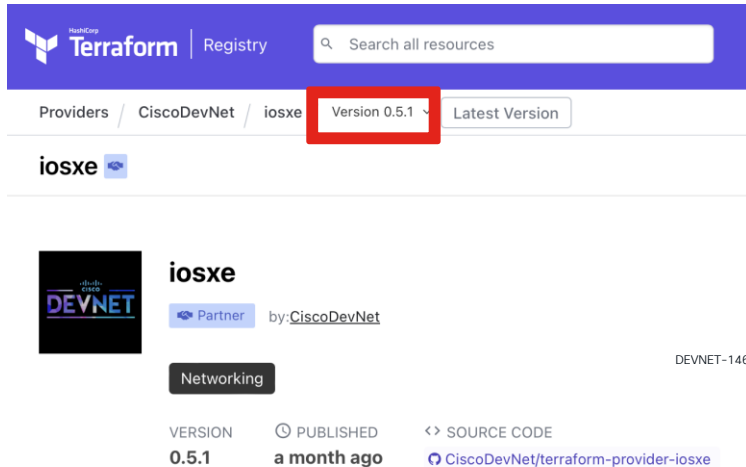
<https://developer.cisco.com/iac/#:~:text=Adopting%20Infrastructure%20as%20Code%20allows,data%20center%20to%20the%20edge>

# IOS XE Terraform Provider

Documentation and details about the provider are available on the Hashicorp Registry

<https://registry.terraform.io/providers/CiscoDevNet/iosxe/latest>

Source code is in the GitHub Repository: <https://github.com/CiscoDevNet/terraform-provider-iosxe/>



HashiCorp Terraform Registry

Providers / CiscoDevNet / iosxe **Version 0.5.1** Latest Version

**iosxe**

**iosxe** Partner by: CiscoDevNet

Networking

VERSION 0.5.1 PUBLISHED a month ago SOURCE CODE [CiscoDevNet/terraform-provider-iosxe](https://github.com/CiscoDevNet/terraform-provider-iosxe/)

## iosxe provider

### > Guides

### > AAA

### > BFD

### > BGP

### > CTS

### > Crypto

### > EVPN

### > General

### > Interface

### > MDT

### > MPLS

### > Management

### > Multicast

### > OSPF

### > QoS

### > Routing

### > Switching

### > System

### > VRF

## General

### Resources

iosxe\_cli

iosxe\_restconf

iosxe\_save\_config

### Data Sources

iosxe\_restconf

# Terraform resource utilizing the CLI RPC

<https://registry.terraform.io/providers/CiscoDevNet/iosxe/latest/docs/resources/cli>

## iosxe\_cli (Resource)

This resource is used to configure arbitrary CLI commands. This should be considered a last resort in case YANG models are not available, as it cannot read the state and therefore cannot reconcile changes.

### Example Usage

```
resource "iosxe_cli" "example" {  
  cli = <<-EOT  
  interface Loopback123  
  description configured-via-restconf-cli  
  EOT  
}
```

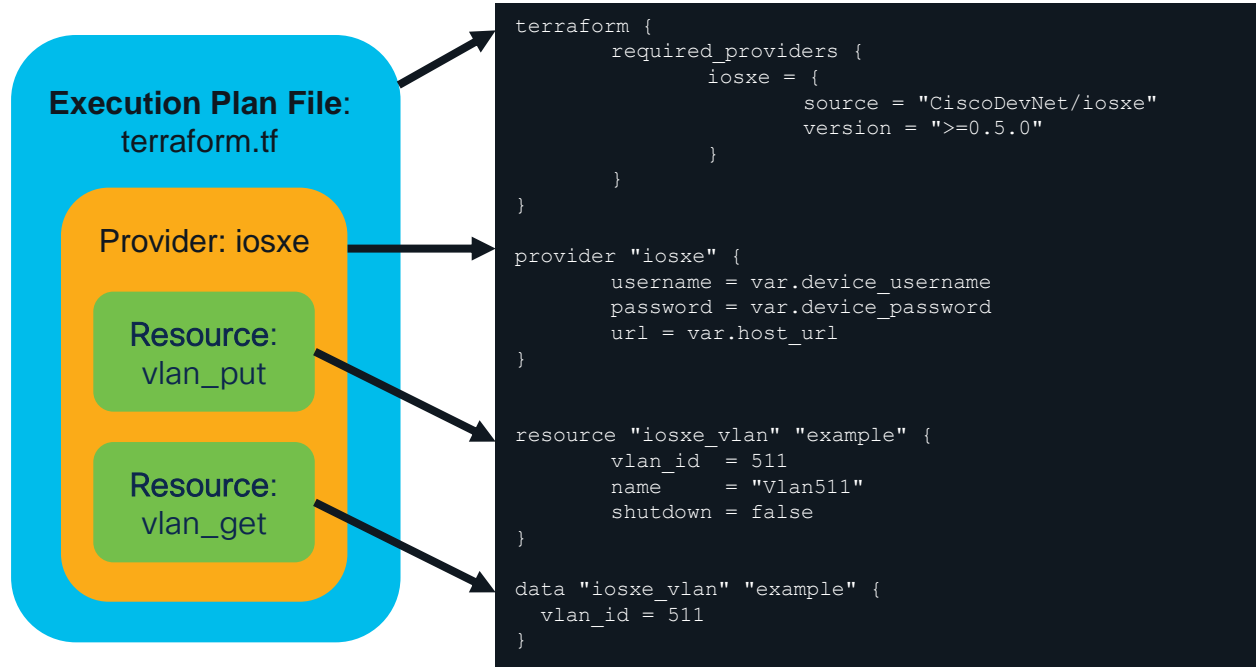
# Terraform Terminology

Terraform uses an execution plan file with a provider and resource definitions

An **execution plan file** defines the provider and resources. It is written in HashiCorp Configuration Language (HCL), similar to JSON, and stored with a .tf extension

A **provider** is a plugin to make a collection of resources accessible

A **resource** (or infrastructure resource) describes one or more infrastructure objects managed by Terraform. With the IOS XE Terraform provider, resources can be considered the same as a configurable feature



# Terraform for Model Driven Telemetry

Enable gRPC Dial-Out telemetry subscriptions for the POE and Basic Device Monitoring use cases

```
aut@pod22-xelab:~/ml_examples$ terraform plan -var-file="var.tfvars"
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# iosxe_mdt_subscription.cpu_subs["100"] will be created
+ resource "iosxe_mdt_subscription" "cpu_subs" {
  + encoding      = "encode-kvgpb"
  + filter_xpath  = "/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds"
  + id           = (known after apply)
  + receivers    = [
    + {
      + address = "10.1.1.3"
      + port    = 57500
      + protocol = "grpc-tcp"
    },
  ]
  + stream       = "yang-push"
  + subscription_id = 100
  + update_policy_periodic = 3000
}

# iosxe_mdt_subscription.cpu_subs["101"] will be created
+ resource "iosxe_mdt_subscription" "cpu_subs" {
  + encoding      = "encode-kvgpb"
  + filter_xpath  = "/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/one-minute"
  + id           = (known after apply)
  + receivers    = [
    + {
      + address = "10.1.1.3"
      + port    = 57500
      + protocol = "grpc-tcp"
    },
  ]
  + stream       = "yang-push"
  + subscription_id = 101
  + update_policy_periodic = 3000
}
```

## Lab Guide Steps:

1. Introduction
2. IOS XE CLI pre-req
3. Terraform Install & headers, device variables
4. Variables and config for MDT subscription
5. Terraform configuration for device monitoring
6. TF Workflow: init, plan, apply & destroy
7. Validation with TF and CLI
8. Conclusion

## iosxe\_mdt\_subscription (Resource)

This resource can manage the MDT Subscription configuration.

### Example Usage

```
resource "iosxe_mdt_subscription" "example" {
  subscription_id = 101
  stream         = "yang-notif-native"
  encoding       = "encode-kvgpb"
  source_vrf     = "Mgmt-vrf"
  source_address = "1.2.3.4"
  update_policy_on_change = true
  filter_xpath   = "/ios-events-ios-xe-oper"
  receivers = [
    {
      address = "5.6.7.8"
      port    = 57600
      protocol = "grpc-tcp"
    }
  ]
}
```

[https://registry.terraform.io/providers/CiscoDevNet/iosxe/latest/docs/resources/mdt\\_subscription](https://registry.terraform.io/providers/CiscoDevNet/iosxe/latest/docs/resources/mdt_subscription)

[https://registry.terraform.io/providers/CiscoDevNet/iosxe/latest/docs/data-sources/mdt\\_subscription](https://registry.terraform.io/providers/CiscoDevNet/iosxe/latest/docs/data-sources/mdt_subscription)

<https://github.com/jeremycohoe/cisco-ios-xe-panda-lab-terraform>

# Use the Docker Container with the Terraform files included!

1. Ensure Docker is installed
2. Run the following commands
  1. `docker pull jeremycohoe/tig_mdt`
  2. `docker run -ti -p 3000:3000 -p 57500:57500 jeremycohoe/tig_mdt`
3. Identify the container ID
  1. `docker ps`

```
sdeweese@SDEWEESE-M-C20V ~ % docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
288Fa40fd38b	jeremycohoe/tig_mdt	"/start -d"	43 seconds ago	Up 42 seconds	0.0.0.0:3000->3000/tcp, 0.0.0.0:57500->57500/tcp	inspiring_gould

4. Enter into the container
  1. `docker exec -it CONTAINER_ID /bin/bash`
  1. Note: replace `CONTAINER_ID` with the ID found in step 3

5. Navigate to the correct folder

1. `cd`
2. `cd cisco-ios-xe-panda-lab-terraform`

6. Modify the device credentials in the header.tf file

```
provider "iosxe" {  
    username = "admin" ← replace with Cisco IOS XE device username  
    password = "XXXXXXXX" ← replace with Cisco IOS XE device password  
    url = "https://your-switch-hostname-or-ip" ← replace with Cisco IOS XE device hostname or IP  
}
```

7. Configure the Cisco IOS XE device using Terraform

```
sdeweese@SDEWEESE-M-C20V ~ % docker exec -it 288fa40fd38b /bin/bash  
root@288fa40fd38b:/# cd  
root@288fa40fd38b:~# ls  
cisco-ios-xe-panda-lab-terraform  start.sh  
root@288fa40fd38b:~# cd cisco-ios-xe-panda-lab-terraform  
root@288fa40fd38b:~/cisco-ios-xe-panda-lab-terraform#  
root@288fa40fd38b:~/cisco-ios-xe-panda-lab-terraform#  
root@288fa40fd38b:~/cisco-ios-xe-panda-lab-terraform# ls  
README.md  cpu.tf  header.tf  imgs  terraform.tfstate  terraform.tfstate.backup  terraform.tfvars  xpaths.tf
```

# DEMO - Install docker container and use Terraform files to configure telemetry subscriptions

```
auto@pod27-xelab: ~ (ssh)
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$
auto@pod27-xelab:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
auto@pod27-xelab:~$
auto@pod27-xelab:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
auto@pod27-xelab:~$
auto@pod27-xelab:~$

c9300-pod27# sh run | s tel
telemetry ietf subscription 6041337
 encoding encode-kvgpb
 filter xpath /process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds
 stream yang-push
 update-policy periodic 30000
 receiver ip address 10.1.1.3 57500 protocol grpc-tcp
c9300-pod27#
c9300-pod27#
Oct 16 21:55:56.082: %HA_EM-6-LOG: catchall: show running-config
```



# Ansible is...

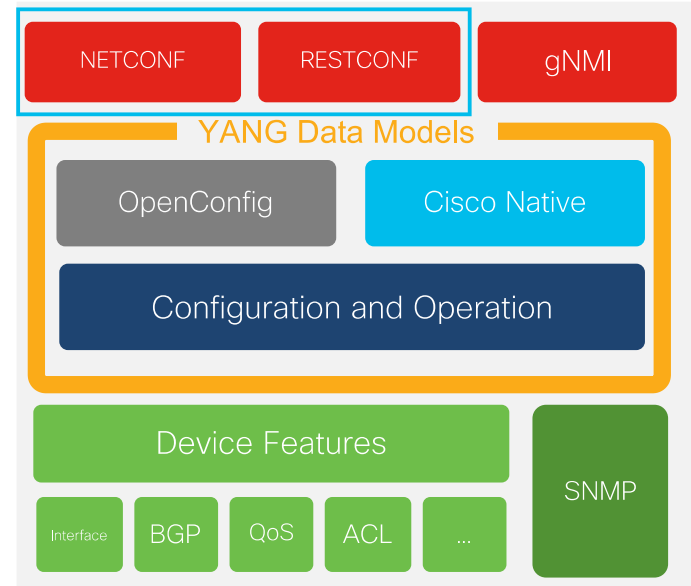


Ansible is an open-source, Infrastructure as Code (IaC) Software suite. It is agentless, meaning there is no installation and no requirements on the target device, other than having an accessible API or interface. Ansible is a state-LESS tooling.

This means, Ansible does NOT look at the current configuration state before making changes.

- minimal in nature and provides a secure and reliable way to interact with remote devices
- highly adaptable and commonly used with other automation tools to accomplish complex workflows.

Ansible can be used with the Cisco IOS XE NETCONF or RESTCONF API



Typically used for device config, *not* infrastructure management

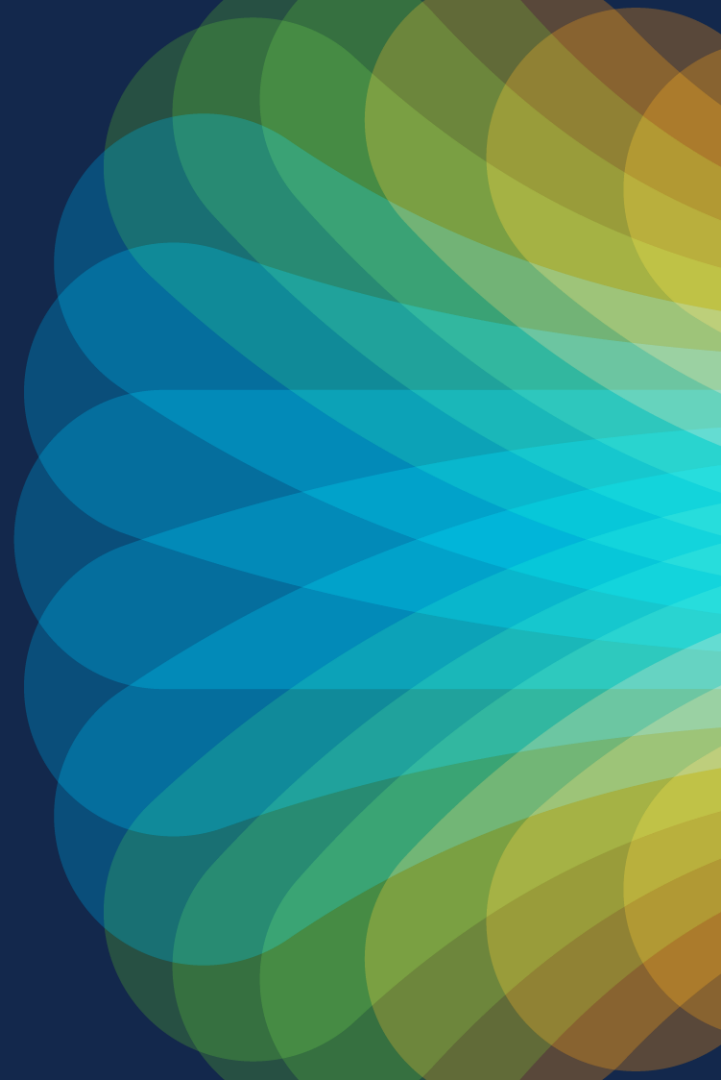
# Example Ansible Playbook

```
1 - name: add telemetry subscriptions
2   hosts: c9300
3   connection: netconf
4   gather_facts: no
5
6   tasks:
7   - name: establish subscription
8     netconf_config:
9       xml: |
10         <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
11           <mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
12             <mdt-subscription>
13               <subscription-id>1</subscription-id>
14               <base>
15                 <stream>yang-push</stream>
16                 <encoding>encode-kvgpb</encoding>
17                 <source-address>10.1.1.5</source-address>
18                 <source-vrf>Mgmt-vrf</source-vrf>
19                 <period>2000</period>
20                 <xpath>/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds</xpath>
21               </base>
22               <mdt-receivers>
23                 <address>10.1.1.3</address>
24                 <port>57500</port>
25                 <protocol>grpc-tcp</protocol>
26               </mdt-receivers>
27             </mdt-subscription>
28           </mdt-config-data>
29         </config>
```

Creates a MDT subscription

[https://github.com/jeremycohoe/ansible-config-samples/blob/master/add\\_sub\\_17.7.yaml](https://github.com/jeremycohoe/ansible-config-samples/blob/master/add_sub_17.7.yaml)

# Resources



# Cisco Catalyst IOS XE Programmability Sessions at Cisco Live Europe Amsterdam 2024

Monday Feb 5

11AM **DEVLIT-2083:**  
Automate Cisco IOS XE Device Configuration Using Terraform (Story)

1:30PM **CISCOU-1024:**  
Model Driven Telemetry with Cisco IOS XE made Simple (Jeremy)

3:30PM **DEVLIT-1965:**  
Automation using multiple API's in the IOS XE Device Programmability Lab (Jeremy)

Session Levels:  
**Beginner**  
**Intermediate**

Tuesday Feb 6

11:30AM **DEVNET-1441:**  
Catalyst 9000 Virtual topology simulation and configuration management solutions (Ama+Jeremy)

11:30AM **DEVLIT-2787:**  
Explore and visualize YANG models with YANG Suite (Story)

1:30PM **DEVVKS-2270:**  
Implementing Cisco IOS XE Model Driven Telemetry with Telegraf, InfluxDB, and Grafana (Jeremy)

1:30PM **CISCOU-2013:**  
Explore Cisco IOS XE Automation Tooling and Use Cases (Story)

4:30PM **DEVLIT-2062:**  
Getting Started with Secure Zero Touch Provisioning (Story)

Wednesday Feb 7

1:30PM **DEVNET-2464:**  
How to Become a Cisco IOS XE Terraform Expert (Story)

4:30PM **CISCOU-2021:**  
gNMI Oh My! Automation with the gRPC micro-services on IOS XE (Jeremy)

Thursday Feb 8

2:30PM **BRKOPS-2455:**  
Infrastructure as Code with Cisco Catalyst 9000 Virtual (Ama+Jeremy)

2:30PM **DEVNET-1468:**  
Programmability, Automation Model Driven Telemetry on Cisco IOS XE with a dash of YANG Suite (Story)

Friday Feb 9

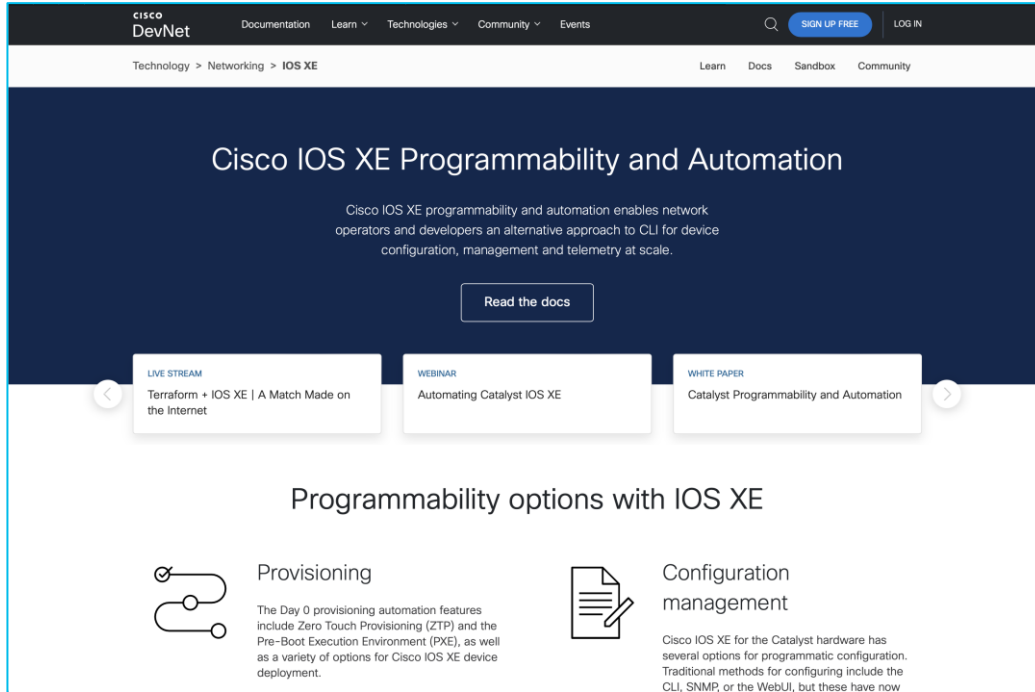


Learn more about sessions this week and on-demand:

<https://blogs.cisco.com/developer/osxeciscoliveemea2024>

# Cisco DevNet Site

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



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

# White Paper – API

Programmability and auto... ^ Q

Table of Contents

Programmability and automatio...-

Day 0: Provisioning automation

Day 1: Model-driven programmability

Day 2: Model-driven telemetry

Day N: Device optimization

Cisco IOS XE operational consistency

Yet Another Next Generation (Y... ^

Day 1: Model-driven program... ^

Tooling: Cisco YANG Suite ^

Day 2: Model-driven telemetry ^

Day N: Device optimization ^

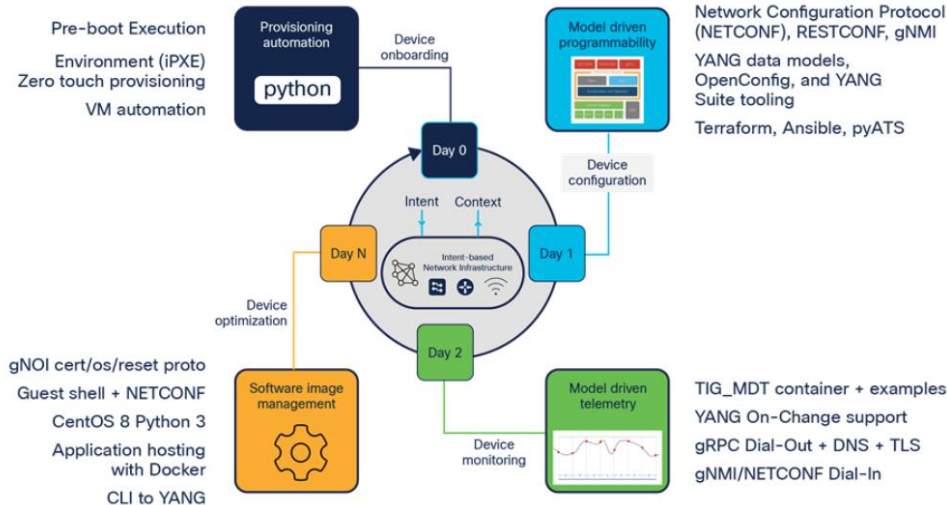
Conclusion

Additional resources ^

Blogs

Products & Services / Switches / Campus LAN Switches - Access / Cisco Catalyst 9300 Series Switches /

## Catalyst Programmability and Automation



<http://cs.co/apiwp>



<http://cs.co/apiwppdf>

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

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

MDT White Paper coming soon!

# 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

## 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

## 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

### 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

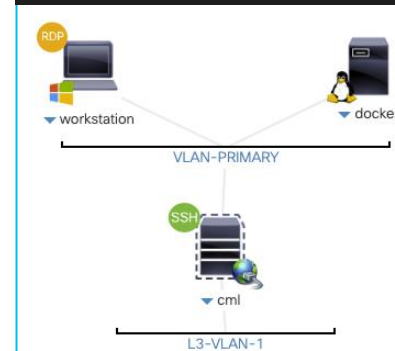
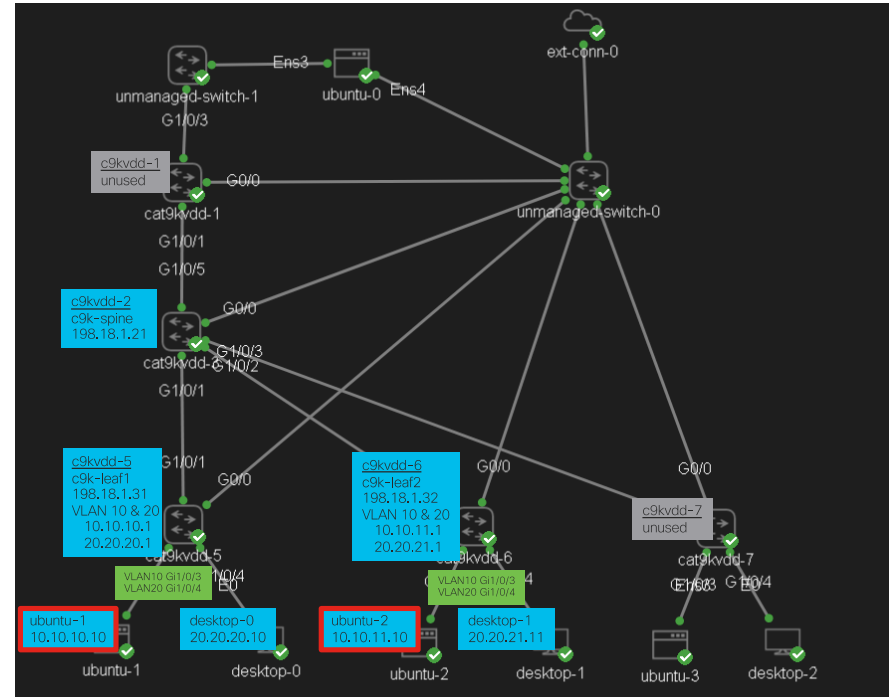
## Bash/PS/Cmd shells

SSH into C9K or Ubuntu

## Postman

## Workspace for EVPN

3x C9K Virtual 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

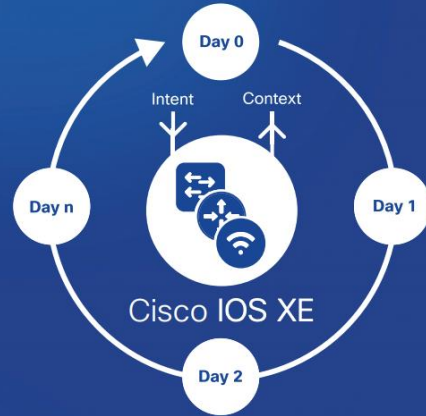
c9kvdd-1 - unconfigured

c9kvdd-7 - unconfigured

# Cisco IOS XE Programmability – Booksprint Book

<http://cs.co/programmabilitybook> OR <https://www.cisco.com/c/dam/en/us/products/collateral/enterprise-networks/nb-06-ios-xe-prog-ebook-cte-en.pdf>

## Cisco IOS XE Programmability Automating Device Lifecycle Management



### Table of Contents

Authors	Telemetry
Acknowledgments	Overview
About this Book	Operational Data
Introduction	Flow Data
Why Programmability Matters	Use Cases
Lifecycle of Network Device Operations	Subscription Tools
Use Cases	Data Collectors
Operational Approaches	Python
Next Steps	Overview
General Concepts	Python WebUI Sandbox
Cisco IOS XE	On-Box Python
What is Programmability?	Advanced On-Box Python
Application Programming Interfaces (APIs)	Common Issues
Programming Languages	Guest Shell
Structured Data	Introduction
Data Encoding Formats	Security
Day 0 Device Onboarding	Configuration and Updates
Introduction	Resource Allocation
Zero-Touch Provisioning (ZTP) Scenarios	Use Cases
Basic ZTP Workflows	Next Steps
Advanced ZTP Workflows	Application Hosting
Considerations	Introduction
Next Steps	Cisco Application-Hosting Framework
YANG	Containers and Virtual Machines
Overview	Use Case
YANG Concepts	Next Steps
YANG Native vs Open Data Models	Controllers
YANG Data Model Highlights	Introduction
YANG Tools	Common Controllers
Network Device APIs	Why Use a Controller?
Overview	DevOps and NetDevOps
NETCONF	Introduction
RESTCONF	Continuous Integration and Delivery
Comparison of NETCONF and	DevOps Tools
RESTCONF	Next Steps
Next Steps	Appendices
	Additional Resources
	Acronyms



# Programmability Configuration Guide

## Book Table of Contents

- Preface
- New and Changed Information
- ✓ Provisioning
  - Zero-Touch Provisioning
  - IPXE
- ✓ Shells and Scripting
  - Guest Shell
  - Python API
  - EEM Python Module
- ✓ Model-Driven Programmability
  - NETCONF Protocol
  - RESTCONF Protocol
  - NETCONF and RESTCONF Service-Level ACLs
  - gNMI Protocol
  - gRPC Network Operations Interface
  - Model Based AAA
  - Model-Driven Telemetry
  - In-Service Model Update
- ✓ Application Hosting
  - Application Hosting
- ✓ OpenFlow
  - OpenFlow
  - High Availability in OpenFlow Mode



[https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1713/b\\_1713\\_programmability\\_cg.html](https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1713/b_1713_programmability_cg.html)

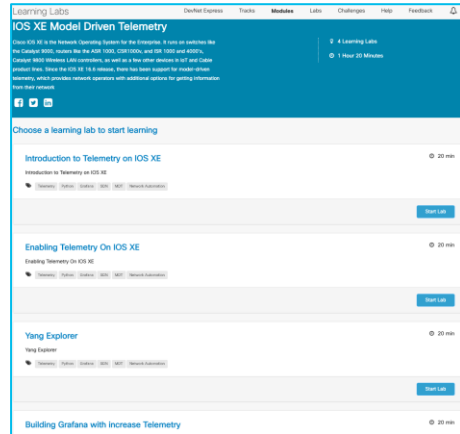
# Learning Lab and Blogs: Telemetry

[https://developer.cisco.com/learning/modules/iosxe\\_telemetry](https://developer.cisco.com/learning/modules/iosxe_telemetry)

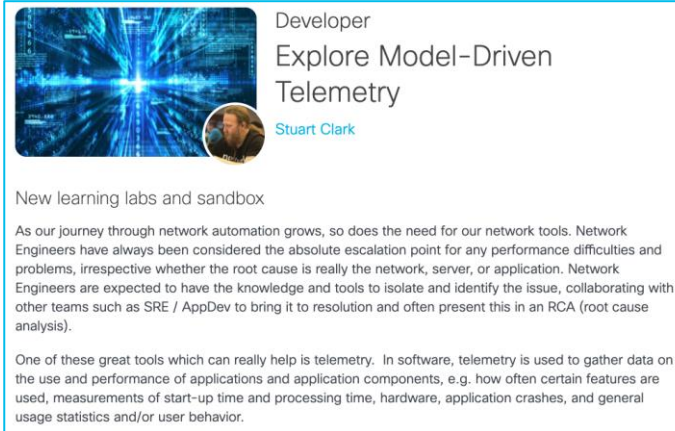
<https://blogs.cisco.com/developer/model-driven-telemetry-sandbox>

<https://blogs.cisco.com/developer/getting-started-with-model-driven-telemetry>

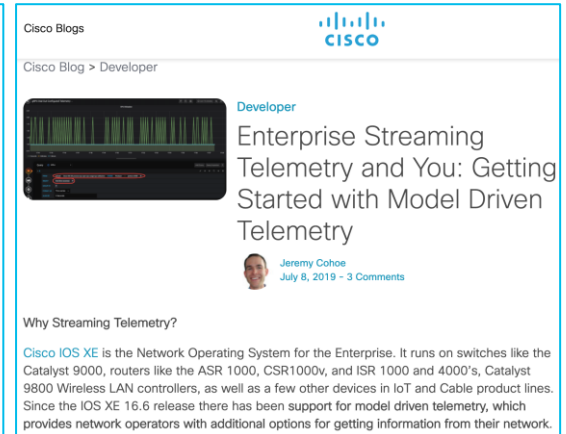
<https://youtu.be/QwwZakkWBng>



The screenshot shows the Cisco Learning Labs interface. At the top, there's a navigation bar with 'Developer Labs' and 'Modules'. The main heading is 'IOS XE Model Driven Telemetry'. Below this, there's a list of learning labs. The first lab is 'Introduction to Telemetry on IOS XE' with a duration of 20 min. The second lab is 'Enabling Telemetry On IOS XE' with a duration of 20 min. The third lab is 'Yang Explorer' with a duration of 20 min. The fourth lab is 'Building Grafana with Increase Telemetry' with a duration of 20 min. Each lab has a 'Start Lab' button.



The blog post features a blue abstract image with a person's profile picture. The title is 'Developer Explore Model-Driven Telemetry' by Stuart Clark. The text reads: 'New learning labs and sandbox. As our journey through network automation grows, so does the need for our network tools. Network Engineers have always been considered the absolute escalation point for any performance difficulties and problems, irrespective whether the root cause is really the network, server, or application. Network Engineers are expected to have the knowledge and tools to isolate and identify the issue, collaborating with other teams such as SRE / AppDev to bring it to resolution and often present this in an RCA (root cause analysis). One of these great tools which can really help is telemetry. In software, telemetry is used to gather data on the use and performance of applications and application components, e.g. how often certain features are used, measurements of start-up time and processing time, hardware, application crashes, and general usage statistics and/or user behavior.'



The blog post is from the Cisco Blogs section, specifically the 'Developer' category. It features a video thumbnail and the title 'Enterprise Streaming Telemetry and You: Getting Started with Model Driven Telemetry' by Jeremy Cohoe, dated July 8, 2019, with 3 comments. The text reads: 'Why Streaming Telemetry? Cisco IOS XE is the Network Operating System for the Enterprise. It runs on switches like the Catalyst 9000, routers like the ASR 1000, CSR1000v, and ISR 1000 and 4000's, Catalyst 9800 Wireless LAN controllers, as well as a few other devices in IoT and Cable product lines. Since the IOS XE 16.6 release there has been support for model driven telemetry, which provides network operators with additional options for getting information from their network.'



The bridge to possible

# Thank you

CISCO *Live!*

The background features a vibrant, multi-colored abstract design. On the left, there are horizontal, wavy bands of color in shades of red, orange, yellow, and green. On the right, a bright white light source emits a series of sharp, radiating lines in various colors, including blue, green, and yellow, creating a sunburst effect.

cisco *Live!*

Let's go