

Catalyst Programmability and Automation

October 2022

Contents

Programmability and automation overview	3
Day 0: Provisioning automation	4
Day 1: Model-driven programmability	4
Day 2: Model-driven telemetry	5
Day N: Device optimization	5
Cisco IOS XE operational consistency	6
Yet Another Next Generation (YANG) data modeling language (RFC 6020, RFC 7950)	6
YANG on GitHub	8
YANG version 1.1 transition	9
YANG Suite tooling	9
Cisco native YANG	10
Standards-based and third-party YANG	15
YANG summary	16
Day 1: Model-driven programmability	16
Security and authentication	16
NETCONF protocol (RFC 6241)	19
RESTCONF (RFC 8040)	23
gNMI	25
Tooling: Cisco YANG Suite	29
Tooling: Cisco pyATS	45
Tooling: Cisco Network Services Orchestrator (NSO)	46
Tooling: Ansible	47
Tooling: Terraform	54
Day 2: Model-driven telemetry	58
NETCONF and SNMP event streams	58
NETCONF dial-in model-driven telemetry	62
gNMI dial-in model-driven telemetry	64
gRPC dial-out model-driven telemetry	64
Tooling: Cisco YANG Suite	69
Tooling: Cisco Crosswork	72
Tooling: Cisco Telemetry Broker	73
Tooling: The TIG stack	74
Day N: Device optimization	76
gNOI	77
CLI to YANG	80
Guest Shell	81
Conclusion	90
Additional resources	90
Developer community and feedback	91
Blogs	91

Programmability and automation overview

The world of programmability has been evolving for years, and with the latest Cisco IOS XE releases, we've included new Yet Another Next Generation (YANG) models to bring additional automation to wireless technology. With the use of APIs, interacting with devices and retrieving data has gotten much easier. Back in the day, we used many commands sent from command-line interfaces (CLIs) to communicate with the software. In addition, the Simple Network Management Protocol (SNMP) was frequently used for network management. Fast forward to today: We now have a new way to interact with software, commonly called an application programming interface, or API. Even though CLIs and SNMP are widely used, they are not as efficient and scalable as APIs.

This document dives into the different programmable interfaces used to communicate with the various Cisco IOS XE devices, specifically the Cisco Catalyst 9300 Series Switches but also the Catalyst 9800 Series Wireless Controllers and the Catalyst 8000 Edge Platforms Family. We discuss the pros and cons of using Network Configuration Protocol (NETCONF), Representational State Transfer Configuration (RESTCONF), and the gRPC network management interface/Google remote procedure call (gNMI/gRPC) protocols, and the main differences between them.

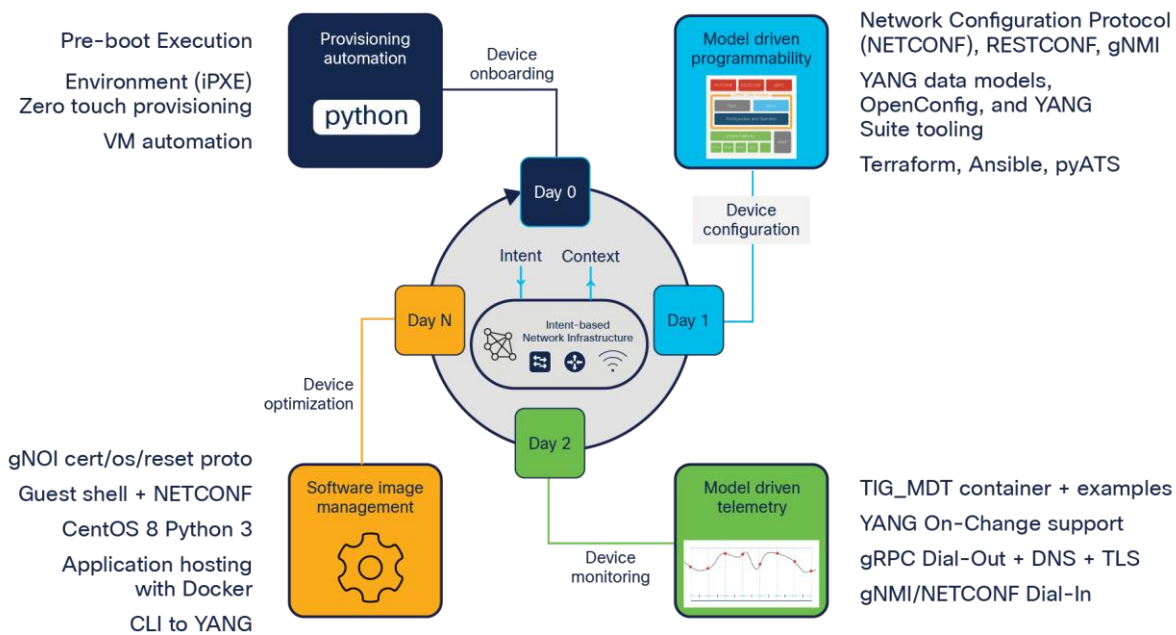


Figure 1.
Programmability and automation overview

Day 0: Provisioning automation

Day 0 provisioning automation features include zero-touch provisioning (ZTP) and the Pre-Boot Execution Environment (PXE), as well as a variety of options for deployment of virtual machines. These day 0 features can be used with a variety of other day 1, day 2, or day N features to achieve provisioning automation in order to successfully configure and deploy new network devices in various campus and enterprise environments.

Day 1: Model-driven programmability

Cisco IOS XE for the Catalyst hardware has several options for programmatic configuration. Traditional methods for configuring include the CLI, SNMP, or the WebUI, but these have now been expanded to include the programmatic interfaces, such as NETCONF, RESTCONF, and the gNMI programmatic interfaces and protocols. YANG data models define what data is accessible over the programmatic interfaces, and they come in several varieties, including Cisco IOS XE features. They are defined within the native data models, while standard and vendor-agnostic features are defined within the open data models. Either model can be used for many tasks. However, features specific to Cisco IOS XE are available only in the native models, which are models created by Cisco specifically for devices and software. The native data models provide the most comprehensive and operational coverage for device functionality.

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

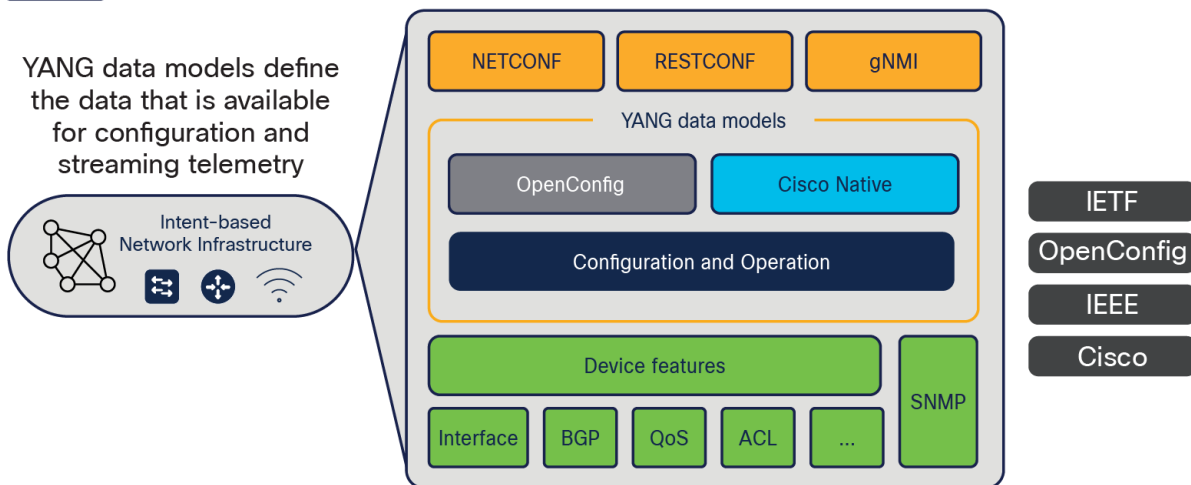
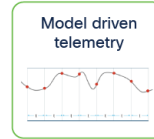


Figure 2.
Model-driven programmable interfaces

Day 2: Model-driven telemetry

Model-driven telemetry or “streaming telemetry” is a day 2 feature using NETCONF and gNMI “dial-in” and gRPC “dial-out” telemetry interfaces. These telemetry interfaces provide a variety of options for publishing network telemetry data to third-party collectors for event processing, analysis, and alerting.



- ↔ 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)

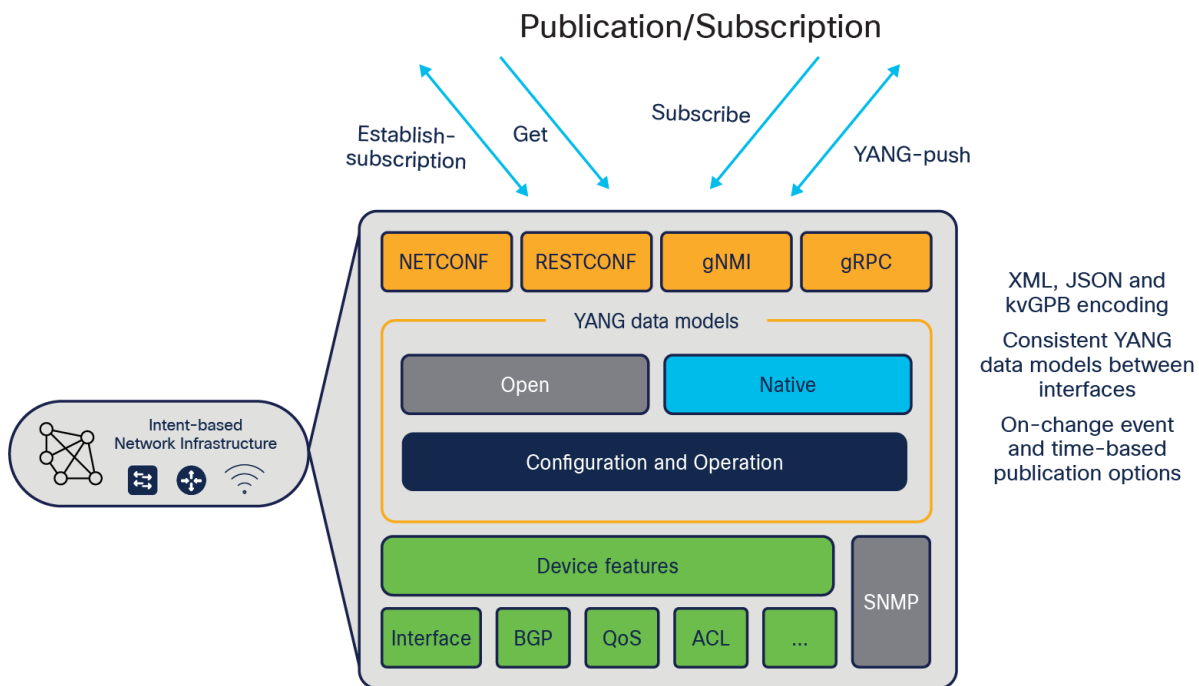


Figure 3.
Model-driven telemetry interfaces

Day N: Device optimization

The day N device optimization features help with operational efficiency, including support for third-party services and integrations within the embedded Guest Shell Linux operating system, the On-Box Python interpreter, and API to Cisco IOS XE. Additionally, device optimization features provide a variety of gNOI workflow APIs that support single call operations for certificate management, operating system software management, and factory reset operations.

Cisco IOS XE operational consistency

Cisco IOS XE software runs on a large variety of hardware solutions, including virtual machines, and the cross-reference of which feature is supported on which platform and the version supported are detailed in the chart below. Additional details for each feature, including the supported hardware and software platforms, are also described in the Programmability Configuration Guide listed in the resources at the end of this document.

	Switching								Wireless				Routing						IOT				SPAG		Cable					
	CAT 3850/3850	CAT 9200L	CAT 9200	CAT 9300L	CAT 9300/9400	CAT 9500	CAT 9500H	CAT 9600	CAT 9100-EWC	CAT 9800-CL	CAT 9800-L	CAT 9800-40/80	ISR 1000	ISR 4000	CR3000	CSR 1000v	CBKV	ASR 1000 Fixed	ASR 1000 Modular	CR500 / CR500L	IR 1100	ESR 6300	IE 3x00	ESS 3300	ASR 900 / 920	NCS 520	NCS 4200	cBR-8		
Provisioning Automation																														
ZTP	16.5+		16.12+	16.12+	16.6+	16.8+	16.12+			17.21	16.12+	16.9+*	16.5+	17.3+	17.2+	17.4+	16.7+	16.8+	17.3+				17.1+	17.1+						
PXE	16.5+		16.9+	16.9+	16.6+	16.8+	16.11+																							
Model Driven Configuration Management																														
NETCONF	16.5+	16.9+	16.9+	16.9+	16.6+	16.8+	16.11+	16.12+	16.10+	16.12+	16.10+	16.8+	16.3+	17.3+	16.3+	17.4+	16.3+	16.3+	17.3+	16.10+	17.1+	16.11+	16.11+	16.8+	16.10+	16.8+	16.8+	16.8+	16.8+	
RESTCONF	16.7+	16.9+	16.9+	16.9+	16.7+	16.8+	16.11+	16.12+	16.11+	16.12+	16.11+	16.8+	16.6+	17.3+	16.6+	17.4+	16.6+	16.6+	17.3+	16.10+	17.1+	16.11+	16.11+	16.8+	16.10+	16.8+	16.8+	16.8+	16.8+	
gNMI		16.12+	16.12+	16.12+	16.8+	16.10+	16.11+		17.6+	17.6+	17.6+	17.8+	17.8+	17.8+		17.8+	17.2	17.2	17.3+	17.2+	17.2+	16.12+	16.12+	17.1	17.1	17.1	17.1	16.12+		
Model Driven Telemetry																														
NETCONF Dial-In	16.6+	16.9+	16.9+	16.9+	16.6+	16.8+	16.11+	16.12					16.8+	16.7+	17.3+	16.10+	17.4+	16.7+	16.8+	17.3+	16.10+	17.1+	16.12+	16.12+	16.9+	16.10+	16.9+	16.9+	16.9+	
gRPC Dial-Out		16.10+	16.10+	16.10+	16.10+	16.10+	16.11+	17.6 ***	17.6 ***	17.6 ***	17.6 ***	16.10+	16.10+	17.3+	16.10+	17.4+	16.10+	16.10+	17.3+	16.10+	17.1+	16.12+	16.12+	16.10+	16.10+	16.10+	16.10+	16.10+	16.10+	
gNMI Dial-In		16.12+	16.12+	16.12+	16.12+	16.12+	16.12+	17.6 ***	17.6 ***	17.6 ***	17.6 ***	17.8+	17.8+	17.8+		17.8+	17.2+	17.2+	17.3+	17.2+	17.2+	17.1+	17.1+	17.1+	17.1+	17.1+	17.1+	16.12+	16.12+	
gNMI explicit wildcard		17.5+	17.5+	17.5+	17.5+	17.5+	17.5+																							
Software Image Management																														
GuestShell (On Box Python)	16.5+ *		16.12+	16.12+	16.6+	16.8+	16.12+			17.21	16.11+	16.9+ **	16.5+	17.3+	16.7+	17.4+	16.7+	16.8+	17.3+				17.1+	17.1+						
gRPC Network Operations Interface (gNOI)																														
cert.proto		17.3+	17.3+	17.3+	17.3+	17.3+	17.3+		17.6+	17.6+	17.6+	17.8+	17.8+	17.8+		17.8+	17.8+	17.8+	17.8	17.8+	17.8+	17.8+	17.8+	17.8+	17.8+	17.8+	17.8+	17.8+	17.8+	17.8+
os.proto			17.5+	17.5+	17.5+	17.5+	17.5+																							
resel.proto		17.7+	17.7+	17.7+	17.7+		17.7+														17.7+	17.7+	17.7+	17.7+						

Figure 4.
Cisco IOS XE operational consistency

Yet Another Next Generation (YANG) data modeling language (RFC 6020, RFC 7950)

YANG is a standards-based data modeling language used to create device configuration requests and retrieve operational (show command) data. It has a structured format similar to that of a human-readable computer program. Several applications are available that can be run on a centralized management platform (for example, a laptop) to create these configuration and operational data requests.

There are both standard (common) YANG data models that apply to all vendors (for example, a request to disable or shut down an Ethernet interface should be identical for both Cisco and third-party devices) as well as device (native, vendor-specific) data models that facilitate configuring or collecting operational data associated with proprietary vendor features.

YANG is a data modelling language for NETCONF, RESTCONF, and gNMI. YANG models within a Cisco IOS XE device have been defined to describe how to structure the data to send or receive. The YANG standard was defined in RFC 6020 and has been updated in RFC 7950. Two main types of YANG models are in use: native and open. The models are further categorized as either configuration or operational models. The configuration models can be used for programmatic configuration, while the operational models can be used with telemetry to show real-time operational data.

OpenConfig is an operator-defined YANG data model
Cisco, IETF, IEEE, etc, are vendor and industry defined

IEEE

IETF

Cisco

OpenConfig

<https://github.com/YangModels/yang>
<https://github.com/openconfig>

Figure 5.

Who defines the YANG models?

There are a variety of YANG modules that can be used for different configuration, operational, and RPC operations, for both programmability and telemetry use cases. The Cisco native data models are the most feature-rich and include support for Cisco features and configurations. IETF and IEEE data models are RFC-defined and ratified YANG standards that have also been implemented. These RFC models support a variety of use cases, from the base streaming telemetry implementation to the listing of support data models, datastores, and streams. OpenConfig is a network operator-driven YANG data model group that also has support for configuration and telemetry use cases.

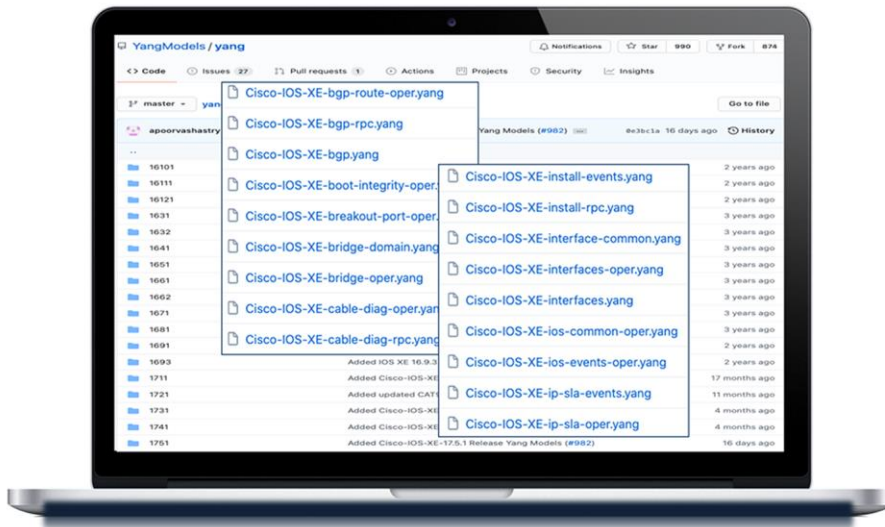
Deviations to YANG are allowed when the server is not able or designed to implement a model as written, and these are specified in the “-deviation.YANG” files provided by the YANG interfaces. These deviations allow the network management station to easily understand which nodes within a particular YANG module are not supported.

YANG on GitHub

The data models are published for each Cisco IOS XE release in the Yang Models repository on GitHub.com at <https://github.com/YangModels/yang/tree/master/vendor/cisco/x>. The supported YANG modules or capabilities for each of the master Cisco IOS XE hardware platforms are also published on GitHub, along with an extensive README, notes on backward-incompatible changes or breaking changes, and a variety of other resources, including both YANG version 1.0 and YANG 1.1 version definitions.

The data models can also be retrieved from the running device over any of the programmatic interfaces by retrieving the contents of the IETF-YANG-Library data model. This will list only the data models supported on the device and can be used programmatically by tooling and systems and network controllers to understand which YANG models are supported on the network device.

Cisco IOS XE - YANG model coverage on GitHub



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

Figure 6.
YANG on GitHub

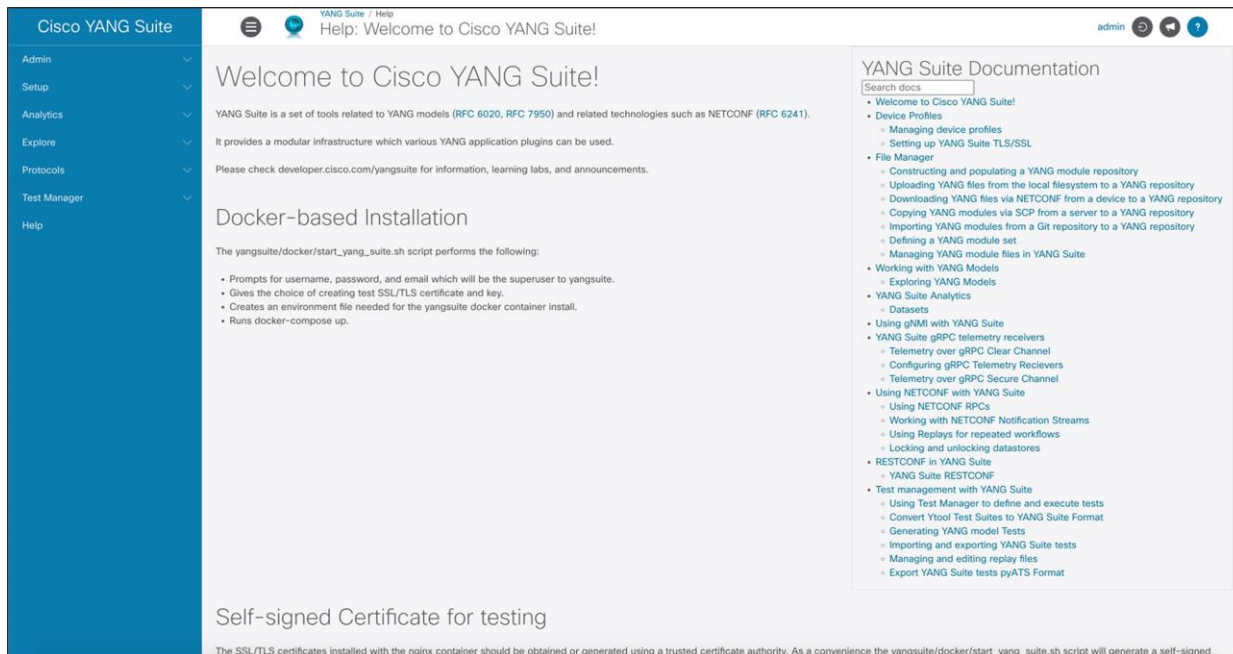
YANG version 1.1 transition

When initially connecting to NETCONF, the “hello” operation returns all supported YANG 1.0 capabilities. However, a GET operation for a data model lists the YANG library or list of supported YANG data models. If the desired application previously parsed the NETCONF "hello" message to retrieve the supported YANG models, the parsing must be modified to reflect how version 1.1 advertises via "ietf-yang-library" instead of the NETCONF "hello" message. This is similar to how RESTCONF behaves, where an HTTP GET to the YANG library data model URI is made to retrieve the list of supported data models.

Cisco native YANG will use YANG 1.0 until Release 17.10, when it will change to YANG 1.1. Customers using YANG 1.0 tooling will need to upgrade to YANG 1.1-compliant tooling—YANG Suite, pyATS, Python, Ansible, and more have a long history of YANG 1.1 support. A simple script has been provided in GitHub to convert models to YANG 1.1 for testing and validation within third-party tooling and integrations. YANG 1.1 is backward compatible with YANG 1.0. (Learn more from IETF here: <https://datatracker.ietf.org/doc/html/rfc7950>)

YANG Suite tooling

[YANG Suite](#) is a tool to easily visualize data models. It can be used to read operational data and configure devices using NETCONF, RESTCONF, gNxl, gRPC, etc.



The screenshot shows the Cisco YANG Suite web interface. The top navigation bar includes the title "Cisco YANG Suite" and a user profile "admin". The left sidebar contains a menu with categories: Admin, Setup, Analytics, Explore, Protocols, Test Manager, and Help. The main content area is titled "Welcome to Cisco YANG Suite!" and includes a sub-header "Docker-based Installation" with a list of steps for installation. Below this, there is a section for "Self-signed Certificate for testing". On the right side, there is a "YANG Suite Documentation" sidebar with a search bar and a list of links to various documentation topics, including "Welcome to Cisco YANG Suite!", "Device Profiles", "File Manager", "Working with YANG Models", and "Test management with YANG Suite".

Cisco native YANG

The Cisco native models are grouped into two main categories: configuration and operational. The configuration modules contain configuration information for the related features, while the operational models provide run-time and operational data about the feature.

Cisco Native YANG: config and oper

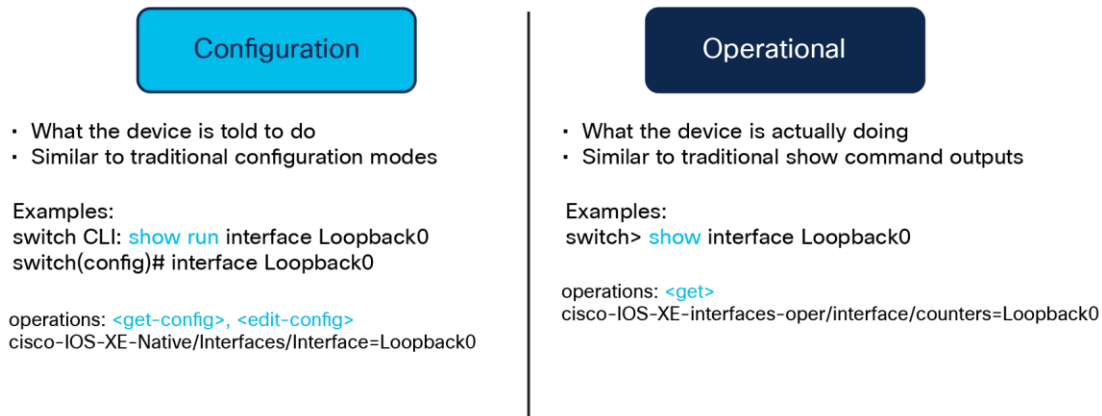


Figure 7.
Cisco native YANG: Configuration and operational models

Cisco native configuration YANG

The Cisco-IOS-XE-native.YANG module is the main native data model for the Cisco IOS XE configuration. This data model has the majority of the “show running configuration” mapped within it. The most commonly used features will have their configuration mapped and modeled here. Common examples include interfaces, VLANs, lines, AAA, and crypto, to name just a few.

The device’s running configuration can be retrieved using this data model, via any of the supported programmable interfaces of NETCONF, RESTCONF, or gNMI. Model-driven telemetry use cases also support this data model, so a subscription to all or some part of this data model results in any changes being streamed or pushed to the remote receiver to support third-party analytics and alerting use cases.

Native YANG examples

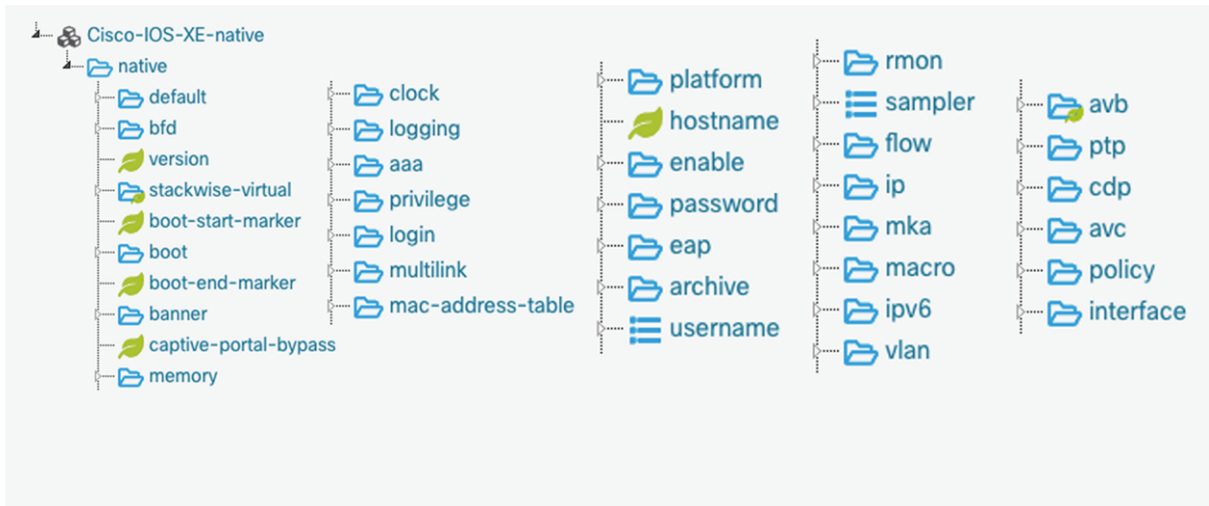


Figure 8.
Native YANG examples

Cisco native operational YANG

Nearly 150 operational or “oper” YANG modules are currently defined, and more are being added with each release, to align with new features and new data that is being mapped and exposed.

The operational models are equivalent to the “show <feature>” command. These do not have configuration data and instead have only operational or run-time data about the feature.

Cisco-IOS-XE-<feature>-oper YANG examples

Feature	YANG Model name	Description
Interface	Cisco-IOS-XE-interface-oper.yang	Interface details and statistics
VLAN	Cisco-IOS-XE-vlan-oper.yang	VLAN details
ACL	Cisco-IOS-XE-ac-oper.yang	Access Control Lists
AAA	Cisco-IOS-XE-aaa-oper.yang	Authentication, Authorization, and Accounting
TCAM / Dataplane Resources	Cisco-IOS-XE-tcam-oper.yang	TCAM utilization per IPv4, IPv6, percent utilization per table
Transceiver	Cisco-IOS-XE-transceiver-oper.yang	Transceiver power levels
Stack	Cisco-IOS-XE-stack-oper.yang	Stack status
Platform	Cisco-IOS-XE-platform-oper.yang	Operating system version, uptime
App-Hosting	Cisco-IOS-XE-app-hosting-oper.yang	Docker application hosting container status

Figure 9.
Cisco-IOS-XE-feature-oper.YANG examples

Cisco native RPC actions YANG

The Cisco-IOS-XE-rpc.yang is one of the data models that can be used with “Other RPC” action within YANG Suite for RPC operations. This data model supports operations for managing files on the flash with “copy” and “delete,” supports license operations, enables reload to be called, and supports release for Dynamic Host Configuration Protocol (DHCP) addresses, among other support actions. These RPC actions are supported on the RFC API interfaces of NETCONF and RESTCONF but are implemented differently on the gRPC interfaces, as described next with gNOI.

Operations supported by the YANG RPCs are similar to those described within the gNOI section: Installing and upgrading operating system software, performing crypto management operations, and doing a factory reset of the device are just a few of the RPC actions that have been modeled. Working with files, the file system, clearing and defaulting features, and debugging and monitoring are also possible with these powerful YANG data models.

Cisco-IOS-XE-rpc

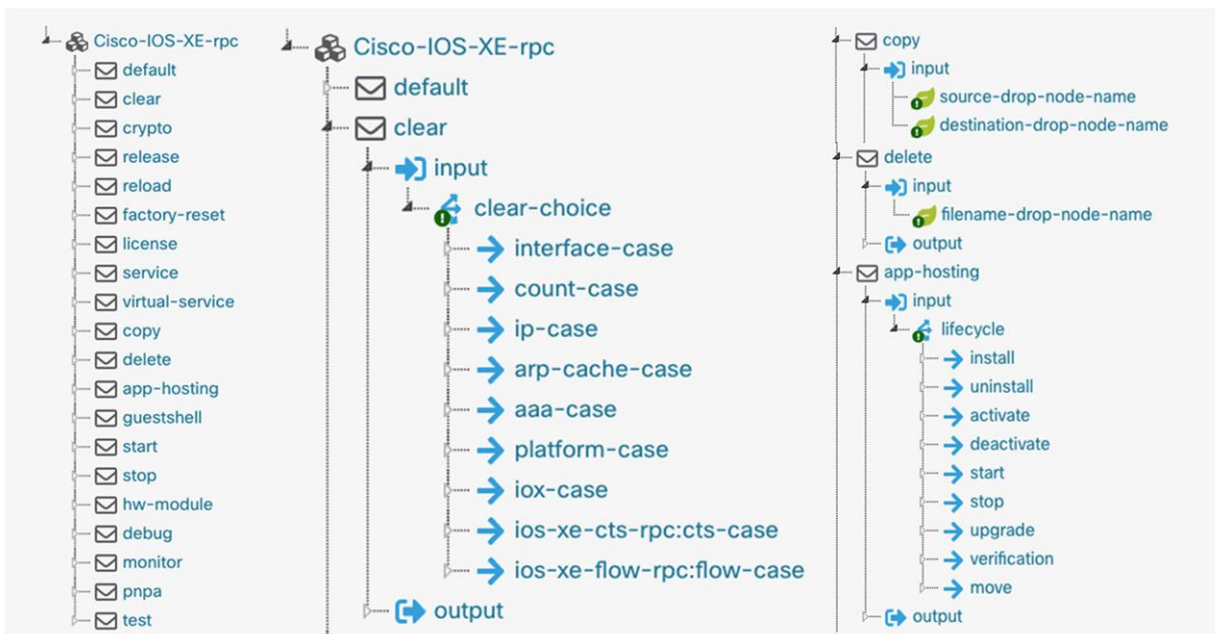


Figure 10.
RPC data model examples

Cisco native “cisco-ia” YANG

Another notable YANG model is “cisco-ia.yang,” short for “Cisco Interface Application,” which enables common RPCs for managing device configuration, specifically “save-config,” which is the programmatic equivalent of the “write memory” or “wr mem” command. Working with NETCONF, time, status of the datastore sync, and actions for configuration revert and rollback are also modeled here.

Cisco IA

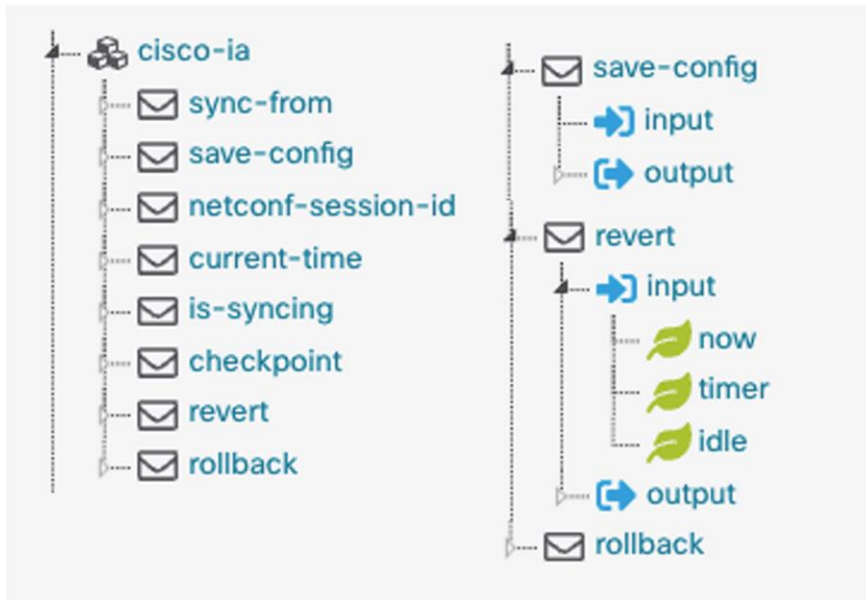


Figure 11.
Cisco IA YANG examples

SNMP MIBs and syslog

The legacy SNMP service and the associated MIBs have been extended to be accessible within the YANG-based APIs. This enables seamless migration from SNMP to YANG by allowing legacy data points to be instrumented using the current YANG APIs. This approach still processes the legacy SNMP objects through the SNMP service and is susceptible to the limitations of this service; therefore, it is not recommended to consume MIBs via YANG interfaces unless necessary. Instead, the YANG data models should be used, which provide more data points and more granular updates.

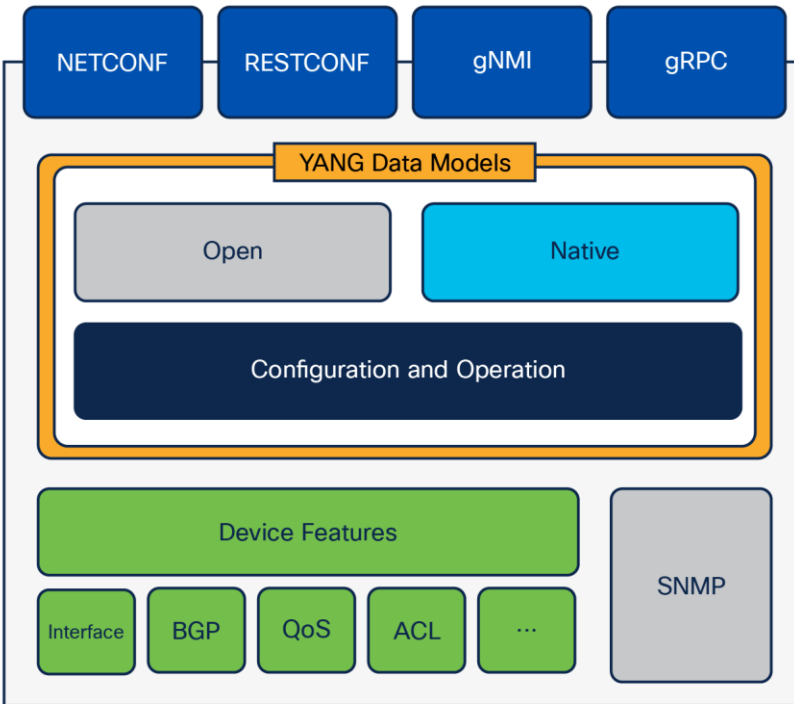


Figure 12.
Programmatic interfaces including SNMP

The syslog extension to SNMP can also be leveraged, which enables syslog messages to be collected from the YANG interfaces. Ensure that the netconf-yang services have the correct SNMP community string so that they can query the SNMP service correctly, as seen in the example below. Creating a subscription using the ietf-event-notifications.YANG model and specifying the stream of “snmpevents” will enable publication of the syslog and other configured events within the NETCONF session.

```
snmp-server community <string> RW
snmp-server enable traps syslog
snmp-server manager
logging history debugging
logging snmp-trap debugging

netconf-yang cisco-ia snmp-community-string <string>
```

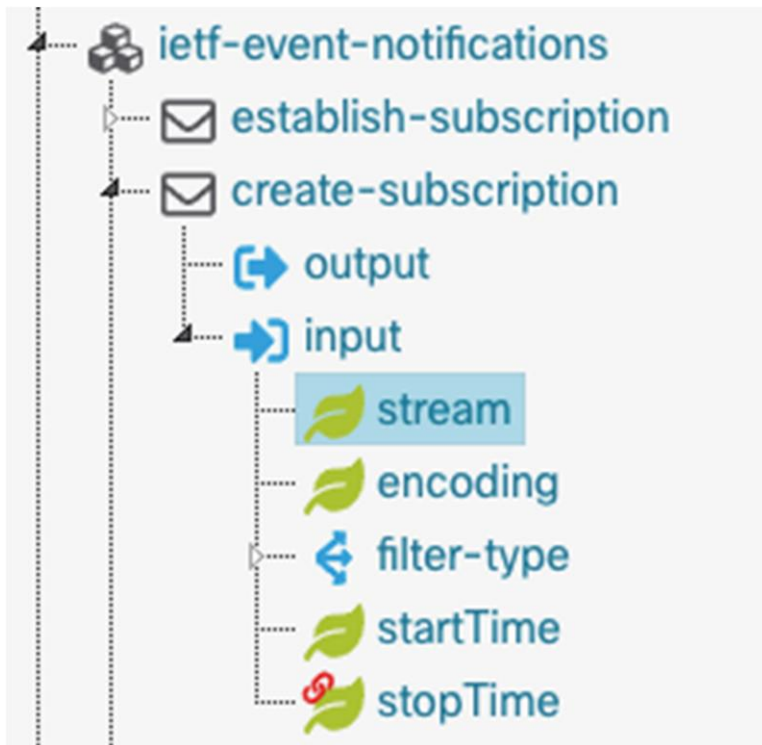


Figure 13.
ietf-event-notifications.YANG tree

Standards-based and third-party YANG

In addition to the Cisco native configuration and operational models, several additional YANG models are supported on the device. The capabilities exchange can be accessed via the YANG interface. There are models for SNMP MIBs, IETF, and OpenConfig. These models can be used in the same way as the native models; however, they offer a limited or subset of the capabilities available on the device.

OpenConfig

OpenConfig is a network operator-driven YANG data model ecosystem that also has support for configuration and telemetry use cases. Unlike the IETF and Cisco native YANG, the OpenConfig (OC) YANG is defined primarily by a working group of network operators and does not include representation from network vendors. As such there are typically no vendor-specific configurations or extensions modeled within OpenConfig, in an effort to make the data model consumable regardless of networking equipment vendor.

The gNMI model-driven programmability and telemetry interface and OpenConfig have a long history, and many network operators who consume OpenConfig YANG may also prefer to leverage the gNMI interface. However, since all YANG data models are advertised for each of the API interfaces of NETCONF, RESTCONF, and gRPC dial-out, the OpenConfig YANG can be used here as well.

IETF

The Internet Engineering Task Force (IETF) YANG data models are the RFC standards that have been ratified and implemented. These include the IETF-yang-push.YANG that is used for telemetry, as well as all other options of NETCONF, such as edit-config; these are defined within the IETF data models as well. There are also some user-facing data models for interface configuration and operational data. While the IETF-interfaces data model can be used to configure and monitor interfaces, Cisco native YANG provides more options and insight into the configuration.

There are also IETF data models related to the NETCONF Access Control Module (NACM), which performs role- and model-based access control (RBAC), and a variety of other RFC standards that are available for use.

YANG summary

YANG data models provide a variety of options to configure, manage, and understand the operational state of the network device. A variety of YANG modules are available, and there is also overlap in coverage between Cisco native, IETF, and OpenConfig, depending on the complexity of the features in use. Cisco native YANG offers the complete range of Cisco IOS XE features that can be managed, while OpenConfig offers a subset of that configuration and contains nothing specific to the vendor or features available in Cisco IOS XE, so it is more of a vendor-neutral model. Regardless of the YANG modules in use, it is common to see a wide variety of YANG types, including Cisco native, OpenConfig, and IETF, all being leveraged as needed to fulfill use cases and business requirements.

Day 1: Model-driven programmability

Day 1 model-driven programmability (MDP or just “programmability”) consists of the various APIs that are available for configuration management, device actions, and operational data retrieval. These APIs use the YANG data modeling language and support a variety of encoding and transportation options, some of which are RFC standards, like NETCONF and RESTCONF, but they also include gNMI.

Security and authentication

There are many considerations when leveraging APIs, and security and authentication are often a major factor in deciding which interface to use and how to securely communicate with it. The Cisco IOS XE APIs support a variety of security and authentication options, including the public key infrastructure (PKI) and SSL/TLS certificates, a combination of username/password and certificate or key-based authentication, and they also support options such as mutual authentication for zero-trust environments where both the client and the server are untrusted and must be validated both ways using certificates.

Support for role-based access control (RBAC), as well as for the RFC6536 NETCONF Access Control Module, enables network operators to secure access to the APIs based on username and user role, and can even be extended to define which YANG data models and which API operations are permitted or denied. This capability gives the API very extensible and granular controls when needed.

Authentication and authorization from upstream TACACS or RADIUS authentication, authorization, and accounting (AAA) services are also supported so that organizations can more centrally manage the users, groups, and roles that are permitted to use the APIs.

NETCONF is the most mature of the programmatic interfaces as it, like RESTCONF, is standards based and has a long history of feature innovation that has been implemented since as early as Cisco IOS XE Release 16.6. NETCONF is the only interface to support the concept of “sessions” that permit the notion of networkwide transactions, including lock/unlock, confirm commit, validate, and a variety of other useful features to ensure that networks are programmed and configured as intended.

	NETCONF	RESTCONF	gNMI
Minimum Cisco IOS XE version	16.6	16.7	16.8
Recommended version	17.9	17.9	17.9
Default TCP port	830	443	9339
Common operations	<get>, <get-config>, <edit-config>, <establish-subscription> <lock> + <confirm-commit>, etc	GET, POST, PUT, PATCH, DELETE, HEAD	Get, Set, Subscribe
Model Language	YANG	YANG	YANG + protobuf
Encoding	XML	XML or JSON	JSON-IETF, protobuf
Security	SSH + PKI certificate/password	HTTP Basic Authentication	TLS certificate with user authentication
Transport protocol	SSH	HTTPS	HTTP/2
Tooling	YANG Suite, ncclient, netconf-console	YANG Suite, postman, curl/python	YANG Suite, gnmi_cli, gnmic

Figure 14. Model-driven programmability interface comparison

AAA configuration example

The most basic and common example, in which the default locally defined users are used for authentication and authorization, is given below. Extending this to RADIUS or TACACS is also a common configuration when a remote authentication and authorization service is used.

```
configure terminal
aaa new-model
aaa authentication login default local
aaa authorization exec default local
exit
```

Username requirements

A user account is required to access the programmatic interfaces. This could take the form of an existing or preconfigured “username” account because a dedicated NETCONF or RESTCONF account is not required. Alternatively, an existing or preconfigured “admin” account can be used. Authentication via TACACS+ or RADIUS is also supported if the user is granted full or privilege Level 15 rights upon login. To create an additional user account with username “netconf” or “restconf” and password “netconf” or “restconf,” use the following commands when using local authentication:

```
Device(config)# username netconf privilege 15 password 0 netconf
Device(config)# username restconf privilege 15 password 0 restconf
```

NETCONF Access Control Module (NACM) and model-based AAA

The programmatic interfaces support NACM, which is a form of RBAC that is defined in RFC6536. This is commonly referred to as model-based AAA. Use this feature to create rules for users that are logging in over the programmatic interfaces so that access to certain models or functions can be permitted or denied as needed. The YANG-based NACM rules are used instead of how TACACS is used with CLI command authorization.

More details can be found in the “Model-Based AAA” chapter of the Cisco IOS XE Programmability User Guide, here: https://www.cisco.com/c/en/us/td/docs/iosxml/ios/prog/configuration/179/b_179_programmability_cg/m_178_prog_model_based_aaa.html

Read-only RBAC example

Prior to Release 17.5, a user with privilege Level 15 was required for any NETCONF operation. Releases 17.5 and later introduced support for lower-privileged and read-only users. Whenever a lower-privileged user attempts to access information such as username or password, that sensitive data gets masked and is not visible to the user.

To enable the API RBAC, enter the following commands:

```
C9300# configure terminal
C9300(config)# username priv1 privilege 1 password netconf
C9300(config)# end
C9300# request platform software yang-management nacm populate-read-rules privilege 1
```

To get configurations as a priv1 user when the NACM rules are populated, use the following command:

```
netconf-console --host 10.0.0.237 --port 830 -u priv1 -p netconf --get-config
```

```
Desktop % netconf-console --host 10.0.0.237 --port 830 -u priv1 -p priv1 --get-config
<?xml version='1.0' encoding='UTF-8'?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <app-hosting-cfg-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-app-hosting-cfg">
    <apps>
      <app>
        <application-name>guestshell</application-name>
        <application-network-resource>
          <management-interface-name>1</management-interface-name>
        </application-network-resource>
      </app>
      <app>
        <application-name>GuestShellApp</application-name>
        <application-network-resource>
          <management-interface-name>0</management-interface-name>
        </application-network-resource>
      </app>
    </apps>
  </app-hosting-cfg-data>
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <version>17.5</version>
```

When specifying the NETCONF read-only privilege level:

- Allowed RPCs are set as: get, get-config, get-schema.
- Sensitive information is masked: Native/enable, native/aaa, native/username.
- Read-only access to all models is provided.

NETCONF protocol (RFC 6241)

NETCONF is a protocol defined by the IETF to “install, manipulate, and delete the configuration of network devices.” NETCONF operations are realized on top of a remote procedure call (RPC) layer using XML encoding and provide a basic set of operations to edit and query the configuration and operational state on a network device.

NETCONF-YANG uses the Cisco IOS Secure Shell (SSH) and can be configured to use Rivest, Shamir, and Adelman (RSA) public keys to authenticate users as an alternative to password-based authentication.

For public-key authentication to work on NETCONF-YANG, the Cisco IOS SSH server must be configured. To authenticate users to the SSH server, use one of the RSA keys configured, by using the “ip ssh pubkey-chain” and user commands.

NACM is a group-based access control mechanism. When users are authenticated, they are automatically placed in an NACM privilege group based on their configured privilege level. Users can also be manually placed in other user-defined groups. The default privilege level is 1. There are 16 privilege levels, PRIV00 to PRIV15.

If a user authenticates via the public key but does not have a corresponding AAA configuration, the user is rejected. If a user authenticates via a public key but the AAA configuration for NETCONF is using an AAA source other than the local one, the user is also rejected. Local and TACACS+ AAA authorization is supported.

An example of how to allow local login in the console line, and to allow NETCONF to authenticate and get authorization from TACACS, can be found here: <https://github.com/jeremycohoe/netconf-tacacs-aaa>

NETCONF feature enablement

To enable the NETCONF interface, enter the following commands:

```
C9300# configure terminal
Device(config)# netconf-yang
Device(config)# exit
```

NETCONF feature status

To verify that the NETCONF interface is operational, the following commands can be used:

```
C9300# show netconf-yang status
```

```
c9300-pod21#show netconf-yang status
netconf-yang: enabled
netconf-yang candidate-datastore: disabled
netconf-yang side-effect-sync: enabled
netconf-yang ssh port: 830
netconf-yang turbocli: disabled

c9300-pod21#
```

```
C9300# show platform software yang-management process
```

Ensure that the NETCONF SSHD “ncsshd” process is running.

```
c9300-pod21#show platform software yang-management process
confd           : Running
nesd            : Running
syncfd         : Running
ncsshd ←       : Running
dmiauthd       : Running
nginx          : Running
ndbmand        : Running
pubd           : Running
gnmib          : Running

c9300-pod21#
```


NETCONF API verification and capabilities exchange

Use SSH to verify that the NETCONF-YANG SSH service is operational when connecting with a SSH client. The capabilities exchange is returned with a list of all supported YANG data models and API capabilities when YANG 1.0 is used. The YANG library is upgraded to version 1.1 starting with Release 17.10.

```
ssh admin@c9300 -p 830
```

```
auto@pod30-xelab:~$ ssh admin@c9300 -p 830
admin@c9300'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>
<capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
```

The NETCONF capabilities exchange can be retrieved by connecting to the device on the default port TCP 830 using SSH. The capabilities exchange lists all available YANG data models supported by the device. Using tools such as YANG Suite, which is detailed in later sections, these YANG modules can be downloaded from the device and analyzed further.

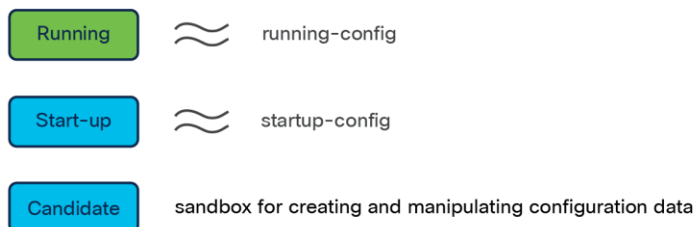
In addition to the capabilities exchange, the `ietf-yang-library` can be used to retrieve the list of supported YANG version 1.0 and YANG version 1.1 data models.

NETCONF datastores

NETCONF has three datastores: running, start, and candidate. The running datastore is the only one required because it contains the current configuration for the device. The start datastore contains the configurations that will be applied to a device on startup. The candidate datastore acts as a sandbox to create configurations before committing them to the running datastore. These datastores follow [RFC 8526](#).

NETCONF Datastore

“A Datastore holds a **copy of the configuration data** that is required to get a device from its initial default state into a desired operational state”



Running is the default and only required Datastore

Figure 15.
NETCONF datastore

NETCONF XML RPC payload examples

The XML RPC payloads below can be generated, modified, and sent from within the Cisco YANG Suite or any other tool that can send XML payloads over the NETCONF interface, such as a Python script. Examples for creating, verifying, and removing a model-driven telemetry (MDT) subscription are listed below. These can be used to create, verify, and remove a WLAN over the NETCONF interface quickly and easily.

Add MDT subscription using XML:

```
<mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
  <mdt-subscription>
    <subscription-id>501</subscription-id>
    <base>
      <stream>yang-push</stream>
      <encoding>encode-kvgpb</encoding>
      <source-address>10.60.0.19</source-address>
      <source-vrf>Mgmt-vrf</source-vrf>
      <period>2000</period>
      <xpath>/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds</xpath>
    </base>
    <mdt-receivers>
      <address>10.12.252.224</address>
      <port>57500</port>
      <protocol>grpc-tcp</protocol>
    </mdt-receivers>
  </mdt-subscription>
</mdt-config-data>
```

Get MDT subscription using XML:

```
<mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
  <mdt-subscription>
    <subscription-id>501</subscription-id>
  </mdt-subscription>
</mdt-config-data>
```

Delete MDT subscription using XML:

```
<mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
  <mdt-subscription xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
nc:operation="delete">
    <subscription-id>501</subscription-id>
  </mdt-subscription>
</mdt-config-data>
```

RESTCONF (RFC 8040)

RESTCONF stands for the HTTP-based Representational State Configuration Protocol ([RFC 8040](#)). It is a stateless protocol that uses the secure HTTP method. RESTCONF uses structured XML or JavaScript Object Notation (JSON) and YANG data models to provide a REST-like API that enables programmatic access to the network device. The RESTCONF API uses the HTTP method and commands such as PUT and GET to send information to and from the Cisco devices. The Catalyst Cisco IOS XE implementation supports the following RESTCONF operations: GET, PATCH, PUT, POST, DELETE, and HEAD.

RESTCONF feature enablement

To enable the RESTCONF interface on the device, enter the following commands:

```
C9300# configure terminal
Device(config)# ip http secure-server
Device(config)# restconf
Device(config)# exit
```

RESTCONF feature status

To verify that the RESTCONF interface is operational, run the command:

```
C9300# show platform software yang-management process
```

Ensure that the “nginx” process is running:

```
c9300-pod21#show platform software yang-management process
confd           : Running
nesd            : Running
syncfd         : Running
ncsshd         : Running
dmiauthd       : Running
nginx ←        : Running
ndbmand        : Running
pubd           : Running
gnmib          : Running

c9300-pod21#
```

RESTCONF verification and encoding

Verify that RESTCONF is running by sending a GET request to the device:

```
auto@pod21-xelab:~$ curl -k -u "developer:Cisco12345" https://sandbox-iosxe-latest-1/restconf/
```

The result will list details from the RESTCONF API in the default encoding of XML. JSON can also be returned by specifying “Accept: application/yang-data+json” in the HTTP headers.

The following example uses the publicly available Cisco IOS XE sandbox:

```
auto@pod21-xelab:~$ curl -k -u "developer:Cisco12345" https://sandbox-iosxe-latest-1/restconf/
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <data/>
  <operations/>
  <yang-library-version>2016-06-21</yang-library-version>
</restconf>
auto@pod21-xelab:~$
```

```
curl -k -u "developer:Cisco12345" https://sandbox-iosxe-latest-1/restconf/
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <data/>
  <operations/>
  <yang-library-version>2016-06-21</yang-library-version>
</restconf>
```

RESTCONF feature verification with JSON

```
curl -H "Accept: application/yang-data+json" -k -u "developer:Cisco12345" https://sandbox-iosxe-latest-1/restconf/
```

```
auto@pod21-xelab:~$
auto@pod21-xelab:~$
auto@pod21-xelab:~$ curl -H "Accept: application/yang-data+json"
-k -u "developer:Cisco12345" https://sandbox-iosxe-latest-1/restc
onf/
{"ietf-restconf:restconf":{"data":{},"operations":{},"yang-librar
y-version":"2016-06-21"}}auto@pod21-xelab:~$
auto@pod21-xelab:~$
auto@pod21-xelab:~$
```

Results in JSON:

```
{"ietf-restconf:restconf":{"data":{},"operations":{},"yang-library-version":"2016-06-21"}}
```

gNMI

gNMI is a gRPC network management interface. gNMI provides a mechanism to install, manipulate, and delete the configuration of network devices, and to view operational data. The content provided through gNMI can be modeled using YANG. gRPC is a remote procedure call developed by Google for low-latency, scalable distributions with mobile clients communicating to a cloud server. gRPC carries gNMI and provides the means to formulate and transmit data and operation requests.

gNMI feature enablement

To enable the gNMI interface, enter the following commands:

```
configure terminal
gnxi
gnxi server
gnxi port 50052
exit
write memory
```

gNMI feature enablement: secure

The following example shows how to enable the gNxi server secure mode.

```
configure terminal
gnxi
gnxi secure-trustpoint trustpoint1
gnxi secure-server
gnxi secure-client-auth
gnxi secure-port 9339
exit
write memory
```

gNMI process status

To verify that the gNMI interface is operational, run the following command:

```
C9300# show gnxi state
```

Ensure that the state is Enabled and the status is Up.

```
c9300-pod21#show gnxi state
State          Status
-----
Enabled        Up
c9300-pod21#
```

More details are available with the “show gnxi state detail” command:

```
C9300# show gnxi state detail
```

```
c9300-pod21#show gnxi state detail
Settings
=====
  Server: Enabled
  Server port: 50052
  Secure server: Disabled
  Secure server port: 9339
  Secure client authentication: Disabled
  Secure trustpoint:
  Secure client trustpoint:
  Secure password authentication: Disabled

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

Full details of the various gNxl microservices are listed with the detail command:

```
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
```

gNMI verification

The YANG Suite tooling can be used to verify and test the gNMI API. This tooling has a gNMI plugin that supports all gNMI operations, including capabilities, get, set, and subscribe. An example of the capabilities operation from the YANG Suite shows that JSON-IETF is supported and provides the gNMI version and a list of the YANG modules that are supported.



gNMI version: 0.7.0

Supported encodings: JSON,JSON_IETF

Model	Version	Organization
Cisco-IOS-XE-aaa-actions-rpc	2021-07-01	Cisco Systems,
Cisco-IOS-XE-aaa-events	2021-07-01	Cisco Systems,
Cisco-IOS-XE-aaa-oper	2021-07-01	Cisco Systems,
Cisco-IOS-XE-aaa-types	2021-07-01	Cisco Systems,
Cisco-IOS-XE-acl-oper	2021-03-01	Cisco Systems,
Cisco-IOS-XE-app-hosting-cfg	2021-07-01	Cisco Systems,
Cisco-IOS-XE-app-hosting-oper	2021-07-01	Cisco Systems,
Cisco-IOS-XE-arp-oper	2021-07-01	Cisco Systems,
Cisco-IOS-XE-ios-common-oper	2021-03-01	Cisco Systems,
Cisco-IOS-XE-bfd-oper	2021-03-01	Cisco Systems,
Cisco-IOS-XE-bgp-common-oper	2019-05-01	Cisco Systems,

Figure 16.
gNMI capabilities

gNMI operations

The following operations are supported by the gNMI interface:

- Capabilities
- gNMI GetRequest
- gNMI SetRequest
- gNMI Subscribe

Also, there are gNOI and gRPC network operations interface workflows that are supported within the gNMI interface and defined in protobuf. These are the certificate management service, the operating system install and upgrade service, and the factory reset service.

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	<code>crypto pki ...</code>	17.3
Os.proto	Network Operating System management	<code>install add file ...</code>	17.5
Reset.proto	Factory Reset and wipe	<code>factory-reset</code> ...	17.7

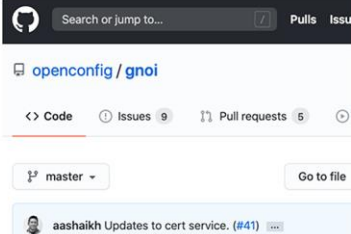


Figure 17.
gRPC network operations interface (gNOI)

gNMI encoding

JSON-IETF is the supported encoding for the gNMI interface, and [RFC 7951](#) defines JSON encoding for YANG data trees and their subtrees. gNMI uses JSON for encoding data in its content layer. The JSON type indicates that the value is encoded as a JSON string. JSON-encoded data must conform to the rules for JSON serialization described in RFC 7951. Both the client and target must support JSON encoding.

Instances of YANG data nodes (leaves, containers, leaf-lists, lists, anydata nodes, and anyxml nodes) are encoded as members of a JSON object or name/value pairs. Encoding rules are identical for all types of data trees, such as configuration data, state data, parameters of RPC operations, actions, and notifications. Every data node instance is encoded as a name/value pair, where the name is formed from the data node identifier. The value depends on the category of the data node. A leaf node has a value, but no children, in a data tree. A leaf instance is encoded as a name/value pair. The value can be a string, number, literal true or false, or the special array [null], depending on the type of the leaf. When the data item at the specified path is a leaf node (which means it has no children and has an associated value), the value of that leaf is encoded directly. (A bare JSON value is included; it does not require a JSON object.) The following example shows a leaf node definition:

```
leaf foo {  
  type uint8;  
}
```

The following is a valid JSON-encoded instance:

```
"foo": 123
```


gNMI wildcard

A gNMI wildcard has been introduced in Release 17.5. Wildcarding is the ability to use a star or wildcard character in a path to match multiple elements. Now it is easier to know which variables are used, instead of having to perform a separate GET request.

There are two types of wildcards, implicit and explicit, and both are supported. GET paths support all types and combinations of path wildcards.

Implicit wildcards: These expand a list of elements in an element tree. An implicit wildcard occurs when a key value is not provided for elements of a list.

Explicit wildcards: These wildcards are directly defined.

Tooling: Cisco YANG Suite

YANG Suite is a tool used for testing and validating the YANG-based APIs on Cisco IOS XE, XR, and NX-OS. YANG Suite was publicly released on GitHub in 2020, and it can be used to generate and send XML payloads to the device over NETCONF, with JSON and RESTCONF, or with gNMI, as well as act as the dial-in and dial-out telemetry receiver. YANG Suite is used extensively by Cisco feature engineering teams and internal and external testing, development and is also widely deployed within campus and enterprise networks as customer tooling.



Figure 18.
Example of NETCONF operation in Cisco YANG Suite

YANG Suite installation

Follow the instructions from the GitHub site at <https://github.com/CiscoDevNet/yangsuite> to complete the installation. Detailed instructions are available for Mac, Windows, and Linux. YANG Suite can be installed as a Docker container or through Python package management. Installing YANG Suite as a Docker container is the recommended method. Additionally, YANG Suite can be installed using pip install.

YANG Suite resources

DevNet landing page: developer.cisco.com/yangsuite

Documentation: developer.cisco.com/docs/yangsuite

Quick start

1. Clone the repository: <https://github.com/CiscoDevNet/yangsuite>
2. Run `start_yang_suite.sh`, or
Run `docker-compose up` if you have already run `start_yang_suite.sh`
3. Access the YANG Suite tool at <https://localhost:8443>

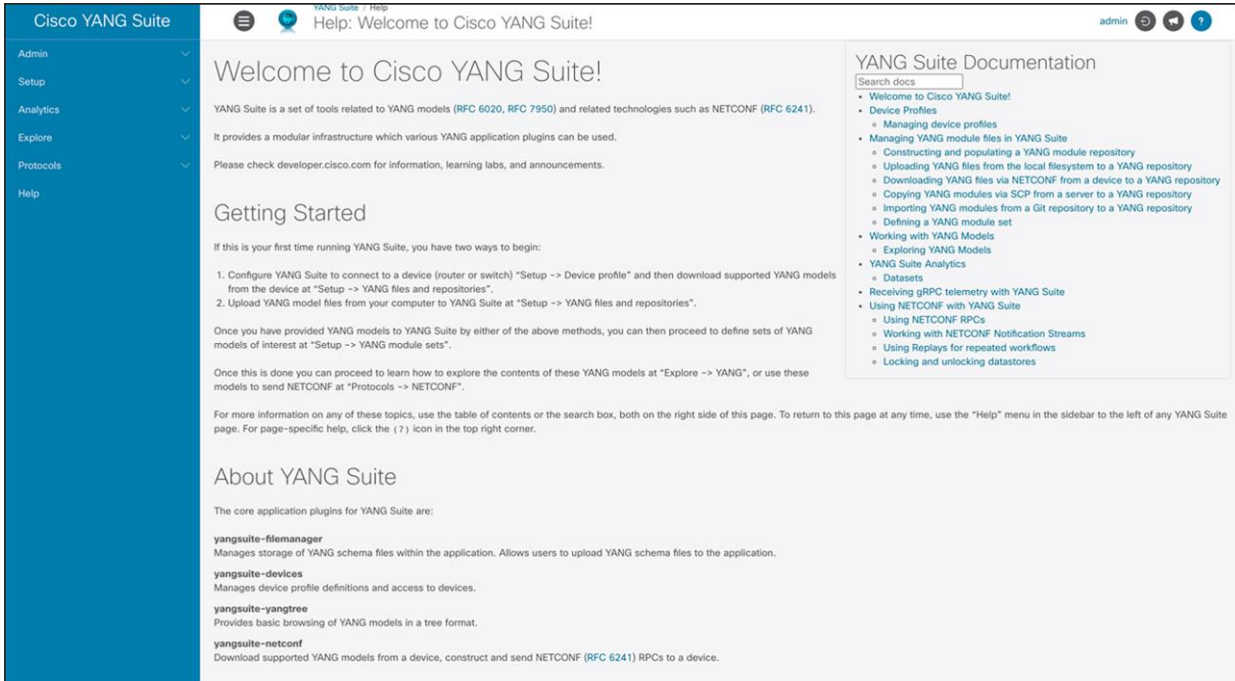
```
git clone https://github.com/CiscoDevNet/yangsuite
cd yangsuite/docker/ ; ./start_yang_suite.sh
or
cd yangsuite/docker/ ; docker compose up
```

When the YANG Suite is ready for use, you will see the following:

```
yangsuite_1 | spawned uWSGI master process (pid: 34)
yangsuite_1 | spawned uWSGI worker 1 (pid: 38, cores: 1)
yangsuite_1 | spawned uWSGI worker 2 (pid: 39, cores: 1)
yangsuite_1 | spawned uWSGI worker 3 (pid: 40, cores: 1)
yangsuite_1 | spawned uWSGI worker 4 (pid: 41, cores: 1)
yangsuite_1 | spawned uWSGI worker 5 (pid: 42, cores: 1)
```

YANG Suite day 0 configuration

When YANG Suite is installed and running, you must complete a few tasks before interacting with the device. Navigate to the Google/Firefox web browser and access YANG Suite at <http://localhost:8443>. Log in using the credentials configured during the installation process. Once logged in, you'll end up at the main YANG Suite application window.



Cisco YANG Suite

Help: Welcome to Cisco YANG Suite!

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 for information, learning labs, and announcements.

Getting Started

If this is your first time running YANG Suite, you have two ways to begin:

1. Configure YANG Suite to connect to a device (router or switch) "Setup -> Device profile" and then download supported YANG models from the device at "Setup -> YANG files and repositories".
2. Upload YANG model files from your computer to YANG Suite at "Setup -> YANG files and repositories".

Once you have provided YANG models to YANG Suite by either of the above methods, you can then proceed to define sets of YANG models of interest at "Setup -> YANG module sets".

Once this is done you can proceed to learn how to explore the contents of these YANG models at "Explore -> YANG", or use these models to send NETCONF at "Protocols -> NETCONF".

For more information on any of these topics, use the table of contents or the search box, both on the right side of this page. To return to this page at any time, use the "Help" menu in the sidebar to the left of any YANG Suite page. For page-specific help, click the (?) icon in the top right corner.

About YANG Suite

The core application plugins for YANG Suite are:

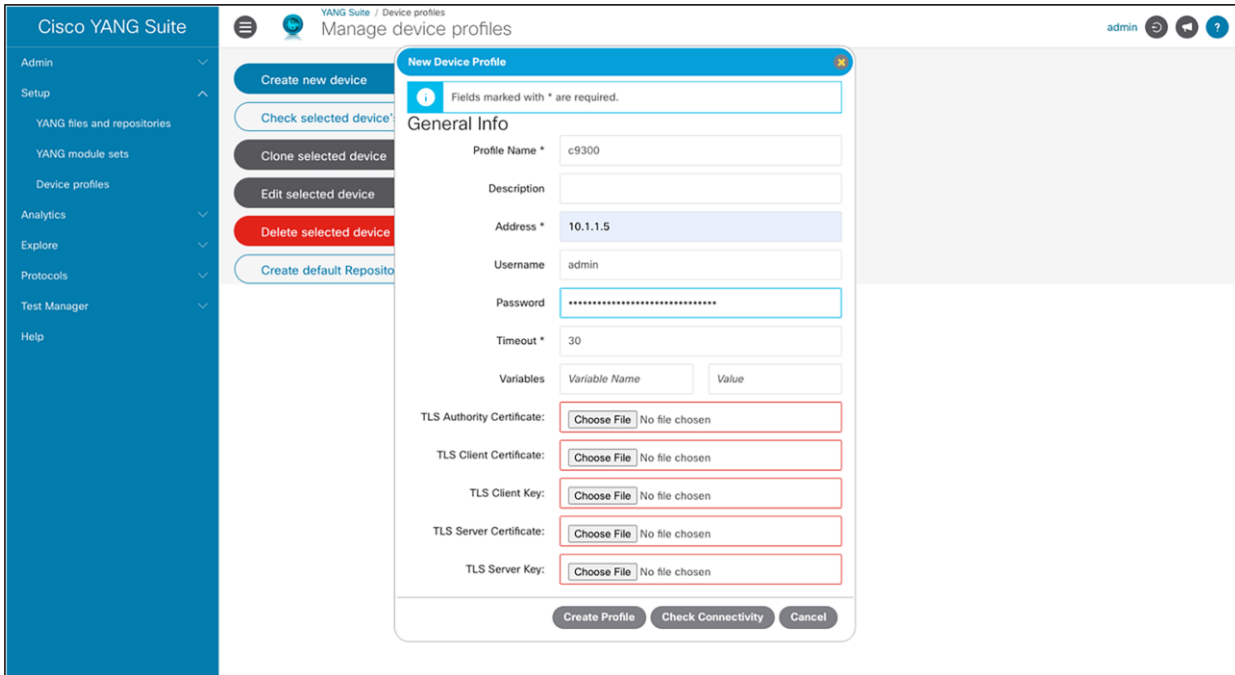
- yangsuite-filemanager**
Manages storage of YANG schema files within the application. Allows users to upload YANG schema files to the application.
- yangsuite-devices**
Manages device profile definitions and access to devices.
- yangsuite-yangtree**
Provides basic browsing of YANG models in a tree format.
- yangsuite-netconf**
Download supported YANG models from a device, construct and send NETCONF (RFC 6241) RPCs to a device.

YANG Suite Documentation

Search docs

- Welcome to Cisco YANG Suite!
- Device Profiles
 - Managing device profiles
- Managing YANG module files in YANG Suite
 - 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
- Working with YANG Models
 - Exploring YANG Models
- YANG Suite Analytics
 - Datasets
- Receiving gRPC telemetry with YANG Suite
- Using NETCONF with YANG Suite
 - Using NETCONF RPCs
 - Working with NETCONF Notification Streams
 - Using Replays for repeated workflows
 - Locking and unlocking datastores

Navigate to the Setup > Device profiles menu. Click "Create new device."



Cisco YANG Suite

Manage device profiles

New Device Profile

Fields marked with * are required.

General Info

Profile Name * c9300

Description

Address * 10.1.1.5

Username admin

Password

Timeout * 30

Variables

Variable Name	Value
---------------	-------

TLS Authority Certificate: No file chosen

TLS Client Certificate: No file chosen

TLS Client Key: No file chosen

TLS Server Certificate: No file chosen

TLS Server Key: No file chosen

The New Device Profile window will pop up, where you must add information about the device. Enter the profile name, address, username, and password. Since YANG Suite now supports gNMI, check the Device supports gNMI box. Check the “Device supports NETCONF,” “Skip SSH key validation for this device,” and “Device supports RESTCONF” boxes. Make sure to enter a username and password for both NETCONF and RESTCONF, if you have configured a separate authentication. Click “Create Profile” to add the device to YANG Suite.

Create New Device Profile

Once the device is added, check the device reachability by clicking “Check selected device’s reachability.” Make sure you see the green checkmark for ping, gNMI, NETCONF, and RESTCONF.

Check Device Connectivity

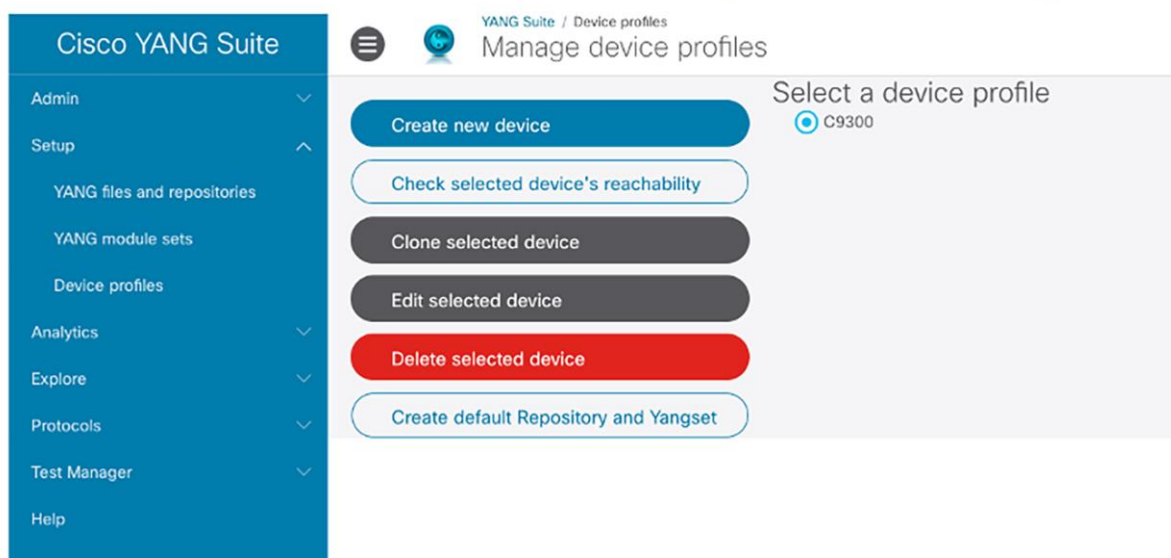
Creating default repository and YANG set

From within YANG Suite, the YANG modules can be downloaded from the device. The YANG repository can be created automatically from the Manage Device Profiles page. The YANG set is a subset of a YANG repository that consists of a set of modules and any other necessary dependencies. A YANG set could store an entire repository, but it's more efficient to narrow the set down to only the models we are interested in. A YANG set needs to be created to build and run RPC(s).

Follow the steps to download the models

On the Manage Device Profiles page, click “Create default Repository and Yangset.” This option will automatically create a default repository and YANG set.

Create Default Repository and Yangset



When the download is completed, the desired modules will appear in the repository and set boxes.

Explore YANG models

1. With the created YANG set, we can easily explore the YANG data models. From the menu on the left side of the page, select Explore > YANG.
2. From the “Select a YANG set” drop-down menu, select a newly created set.
3. In the “Select YANG module(s)” box, enter any data model of choice. In this example, we explore the Cisco-IOS-XE-interfaces-oper module. To find this in the drop-down list, you can start typing keywords such as “native” or “interfaces” to quickly find the correct YANG module.
4. Click the “Load module(s)” button. After a moment, the left column will be populated with a tree view of the contents of the module. Initially the tree view shows only the module itself, but you can click the triangle icon next to it to expand the tree.
5. Refer to the screenshot to examine the structure of the model and its contents.

Explore YANG Models

Two important pieces of YANG model metadata are the xpath and the prefix. These fields are used with streaming telemetry to retrieve information. If a telemetry subscription was to be created based on the Cisco IOS XE interfaces YANG data model, the xpath of “/interfaces/interface/interface-type” and “interfaces-ios-xe-oper” would be used to retrieve and publish information from those models. Learn more about streaming telemetry in the MDT section below.

filter xpath / PREFIX : XPATH

filter xpath /process-cpu-ios-xe-oper:cpu-usage/cpu-usage

Note: For Cisco IOS XE native models, we can simply use the xpath, in this case “/cpu-usage/cpu-usage”.

YANG Suite: NETCONF

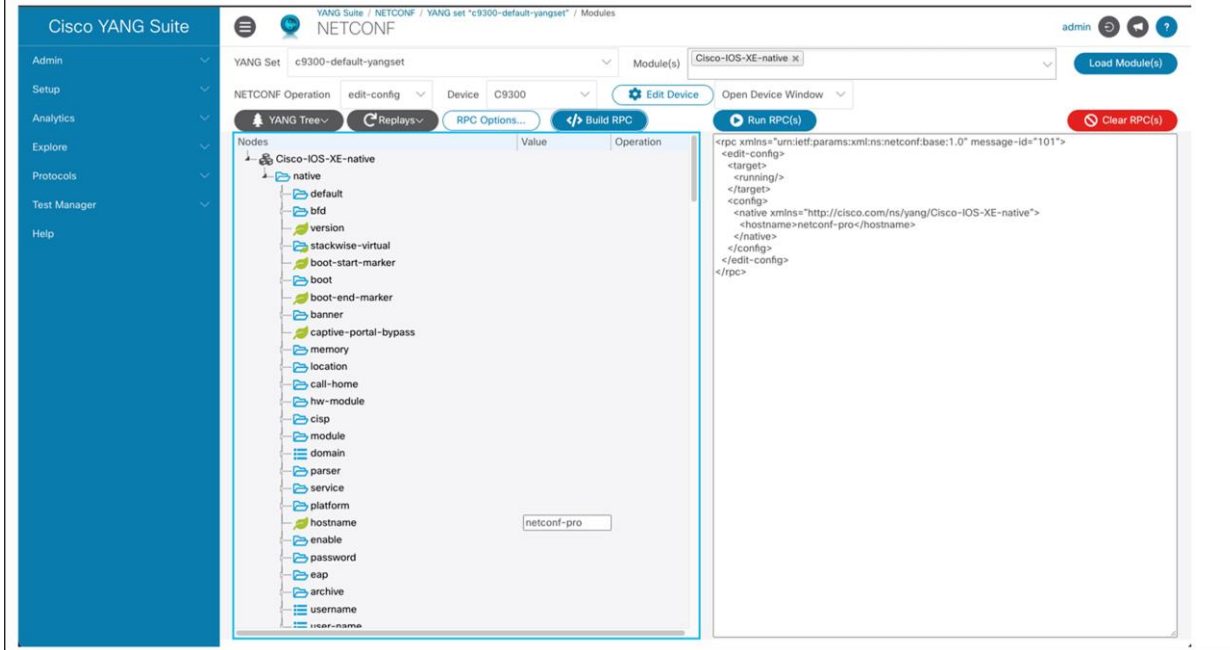
YANG Suite enables interaction with the devices using most of the programmatic interfaces: NETCONF, RESTCONF, gNMI, and gRPC. With the help of the NETCONF operation get-config, it is easy to retrieve all or part of the specified configuration datastore as seen in the screenshot below. Also, the NETCONF operation edit-config loads a specified configuration to a specified target configuration.

Steps to access the NETCONF plugin using YANG Suite

Set hostname on Catalyst 9300 Series switch

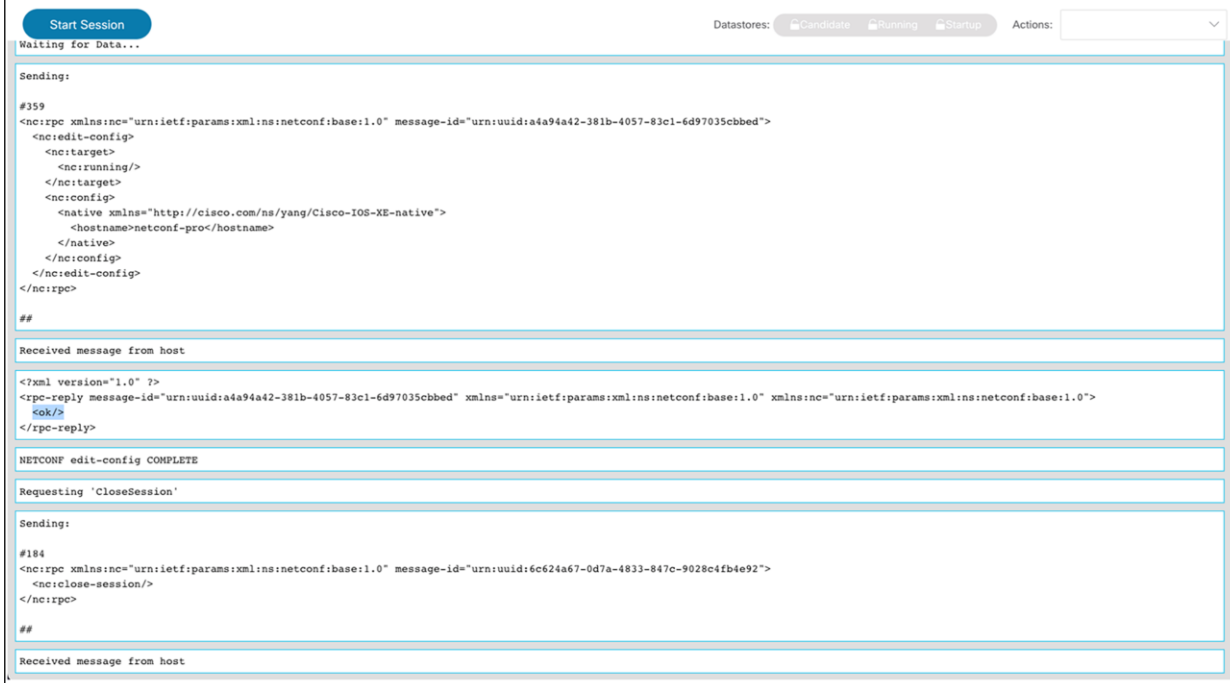
1. Protocol: NETCONF
2. YANG set: c9300-default-yangset
3. Modules: Cisco-IOS-XE-native (Note: Start typing " native" to filter through the options in the drop-down menu.)
4. Click the blue "Load Modules" button.
5. NETCONF operation: edit-config
6. Device: C9300
7. Wait for the tree to appear in the gray box on the left. (Note: If you get an Error 500 popup, just ignore it and close the popup.)
8. Once the YANG tree is created, select " hostname." (Note: Select CONTROL + F or COMMAND + F to find "hostname" on the page.)
9. Select the word "hostname" and add a string to the text field such as "configured-by-a-panda-pro."
10. Click the red "Clear RPCs" button for a fresh start for the next NETCONF payload.
11. Click the blue "Build RPC" button to **generate** the XML RPC that is based on the YANG model and inputs provided. The XML can be reviewed, edited, or used in other tooling or orchestration systems as needed.

Build NETCONF XML Payload to SET Hostname



12. Click the blue “Run RPC(s)” button to **send** the XML RPC to the switch’s NETCONF interface in order to retrieve the configuration as requested.
13. In the new tab that opens, notice the hostname in the response. The device will respond with “<ok/>” if the configuration was applied properly and there were no errors. (**Note:** You may need to scroll to the bottom of the page.)

Device Response after NETCONF SET



Get hostname on Catalyst 9300 Series switch

1. Protocol: NETCONF
2. YANG set: c9300-default-yangset
3. Modules: Cisco-IOS-XE-native (Note: Start typing “native” to filter through the options in the drop-down menu.)
4. Click the blue “Load Modules” button.
5. NETCONF operation: get-config
6. Device: C9300
7. Wait for the tree to appear in the gray box on the left. (Note: If you get an Error 500 popup, just ignore it and close the popup.)
8. Once the YANG tree is created, select "hostname." (Note: Select CONTROL + F or COMMAND + F to find "hostname" on the page.)
9. Select the word "hostname," but there's no need to add anything to the text box (leave it blank for the “get” request).

Build NETCONF XML Payload to GET Hostname

The screenshot shows the Cisco YANG Suite interface. The top navigation bar includes 'Admin', 'Setup', 'Analytics', 'Explore', 'Protocols', 'Test Manager', and 'Help'. The main content area is titled 'NETCONF' and shows the following configuration:

- YANG Set: c9300-default-yangset
- Module(s): Cisco-IOS-XE-native
- NETCONF Operation: get-config
- Device: C9300

The 'Nodes' tree on the left shows the hierarchy of the Cisco-IOS-XE-native module, with 'hostname' selected. The 'Value' field for 'hostname' is empty. The 'Run RPC(s)' button is highlighted in blue. The XML payload on the right is:

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

10. Click the blue “Run RPC(s)” button to **send** the XML RPC to the switch’s NETCONF interface in order to retrieve the configuration as requested.
11. In the new tab that opens, notice the hostname in the response. (Note: You may need to scroll to the bottom of the page.)

Device Response after NETCONF GET

```
Start Session [Candidate] [Running] [Startup] Actions: [v]

Sending:
#336
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:4c8cb5ac-15c6-4fae-83bc-e9f9b1f7ad43">
  <nc:get-config>
    <nc:source>
      <nc:running/>
    </nc:source>
    <nc:filter>
      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <hostname/>
      </native>
    </nc:filter>
  </nc:get-config>
</nc:rpc>

##

Received message from host

<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:4c8cb5ac-15c6-4fae-83bc-e9f9b1f7ad43" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
      <hostname>netconf-proc/hostname</hostname>
    </native>
  </data>
</rpc-reply>

NETCONF get-config COMPLETE

Requesting 'CloseSession'

Sending:
#184
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="urn:uuid:bd42736b-d523-4308-9780-e491928d8b82">
  <nc:close-session/>
</nc:rpc>

##
```

Download a Python script

YANG Suite offers the option to download a Python script to replay a specific operation to view or set configurations on the device. To download a script, build a payload (see examples in “Get Hostname on Catalyst 9300 Series Switch” or “Set Hostname on Catalyst 9300 Series Switch” above). Then click the “Replays” button. In the drop-down menu, select “Generate Python script” and the file will automatically download on the device running YANG Suite, as in the example below.

Generate Python Script

The screenshot shows the YANG Suite interface for NETCONF. The top navigation bar includes 'YANG Suite / NETCONF / YANG set "c9300-default-yangset" / Modules' and a user profile 'admin'. The main area shows 'YANG Set' as 'c9300-default-yangset' and 'Module(s)' as 'Cisco-IOS-XE-native'. The 'NETCONF Operation' is set to 'get-config' and the 'Device' is 'C9300'. A 'Replays' button is active, and a dropdown menu is open, highlighting 'Generate Python script'. The 'Nodes' tree on the left shows the configuration structure, and the right pane displays the XML payload for the get-config operation.

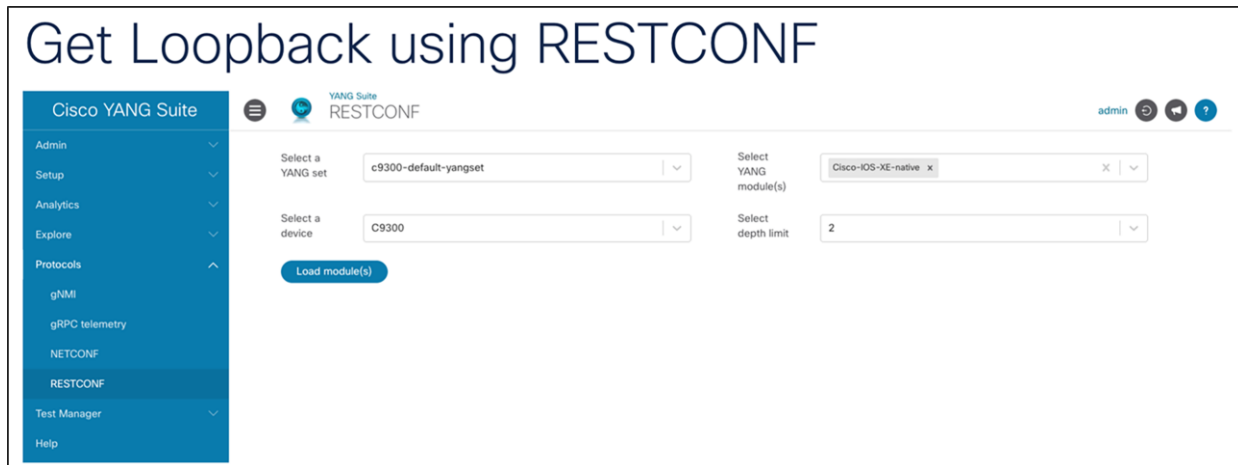
YANG Suite: RESTCONF

To access the RESTCONF plugin using YANG Suite, navigate to Protocols > RESTCONF in the left pane of the YANG Suite application. Make sure to fill out all the necessary fields to load the YANG modules from the device and generate API(s). For the YANG module, we used the Cisco-IOS-XE-native to get all the information about the configured features on the Catalyst 9300 device.

Get hostname on Catalyst 9300 Series switch

1. Protocol: RESTCONF
2. Select a YANG set: c9300-default-yangset
3. Select a device: C9300
4. Select YANG modules: Cisco-IOS-XE-native
5. Select depth limit: 2
6. Click the “Load Module(s)” button.

Get Loopback using RESTCONF



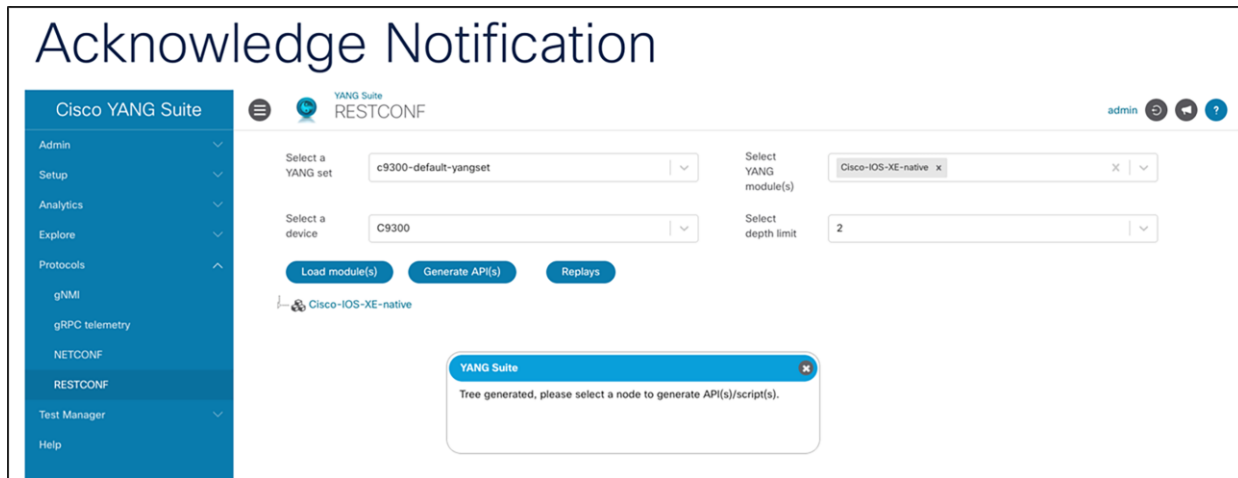
The screenshot shows the Cisco YANG Suite interface for the RESTCONF protocol. The left sidebar is expanded to 'Protocols' > 'RESTCONF'. The main area contains the following configuration fields:

- Select a YANG set: c9300-default-yangset
- Select a device: C9300
- Select YANG module(s): Cisco-IOS-XE-native x
- Select depth limit: 2

A 'Load module(s)' button is visible below the device selection field.

7. Once the tree loads, close the popup that says “Tree generated, please select node(s) to generate API(s).”

Acknowledge Notification



The screenshot shows the Cisco YANG Suite interface after clicking 'Generate API(s)'. The configuration fields are the same as in the previous screenshot. Below the 'Load module(s)' button, there are three buttons: 'Load module(s)', 'Generate API(s)', and 'Replays'. A notification popup is displayed in the center of the screen with the following text:

YANG Suite
Tree generated, please select a node to generate API(s)/script(s).

8. Expand the tree by selecting the arrow next to “Cisco-IOS-XE-native.”
9. Search for “interface” in the expanded tree. (Note: Select CONTROL + F to find " interface" on the page.)
10. Expand “interface” and search for “Loopback.” (Note: Select CONTROL + F to find “Loopback” on the page.)

Select interfaces > Loopback

The screenshot displays the Cisco YANG Suite interface. On the left is a navigation sidebar with categories like Admin, Setup, Analytics, Explore, Protocols, gNMI, gRPC telemetry, NETCONF, RESTCONF, Test Manager, and Help. The main area shows configuration filters: 'Select a YANG set' (c9300-default-yangset), 'Select a device' (C9300), 'Select YANG module(s)' (Cisco-IOS-XE-native), and 'Select depth limit' (2). Below these are buttons for 'Load module(s)', 'Generate API(s)', and 'Replays'. The central tree view shows a hierarchy starting with 'policy' and 'interface'. The 'interface' module is expanded, listing various sub-modules such as AppNav-Compress, ATM, Ethernet, and Loopback. The 'Loopback' module at the bottom is highlighted with a blue selection bar.

11. Click the blue “Generate API(s)” button. (Note: It may take a few moments to load the APIs)
12. Click the blue “Show API(s)” button.

View the Generated RESTCONF APIs

OpenAPI v3.0.3 3.0.3 OAS3

HOST DESTINATION: https://10.1.1.5:443 (proxy through YANG Suite server)

Servers

/restconf/proxy/https://10.1.1.5:443/restconf - YANG SUITE Proxy RESTCONF API

default

PATCH	/data/Cisco-IOS-XE-native:native/interface/Loopback	✓	🔒
PUT	/data/Cisco-IOS-XE-native:native/interface/Loopback	✓	🔒
POST	/data/Cisco-IOS-XE-native:native/interface/Loopback	✓	🔒
GET	/data/Cisco-IOS-XE-native:native/interface/Loopback	✓	🔒
GET	/data/Cisco-IOS-XE-native:native/interface/Loopback={Loopback-name}	✓	🔒
DELETE	/data/Cisco-IOS-XE-native:native/interface/Loopback={Loopback-name}	✓	🔒
PATCH	/data/Cisco-IOS-XE-native:native/interface/Loopback={Loopback-name}/description	✓	🔒
PUT	/data/Cisco-IOS-XE-native:native/interface/Loopback={Loopback-name}/description	✓	🔒

Close

13. Select the link next to the GET operation corresponding to the API call “/data/Cisco-IOS-XE-native:native/interface/Loopback.”
14. Click the "Try it out" button.

Try out the RESTCONF GET Loopback API

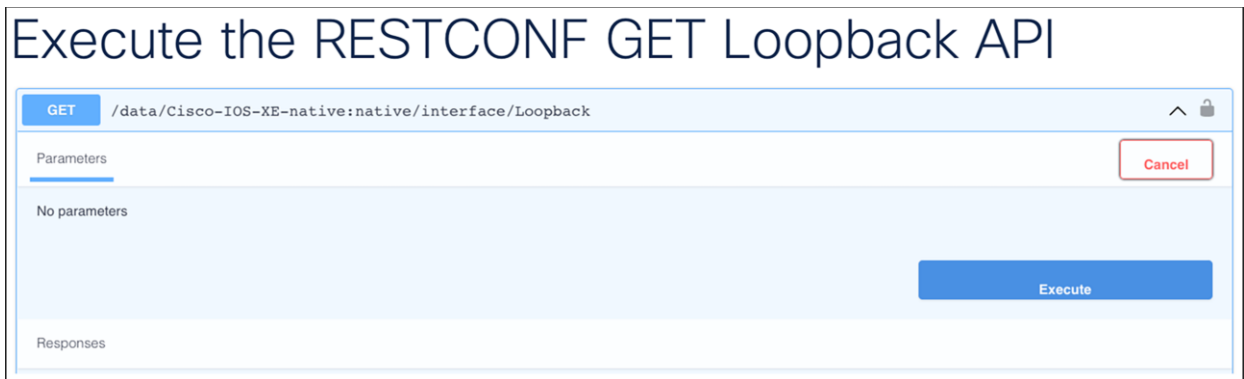


The screenshot shows the RESTCONF GET Loopback API interface. The URL is `/data/Cisco-IOS-XE-native:native/interface/Loopback`. The interface includes a "Parameters" section with "No parameters" and a "Try it out" button. Below is a "Responses" section with a table of status codes and descriptions. Each response has a "Media type" dropdown set to `application/yang-data+json` and a "Links" column with "No links".

Code	Description	Links
200	Successful OK	No links
400	Internal error	No links
405	Method not allowed	No links
500	Internal server error	No links

15. Click the “Execute” button to send the RESTCONF payload and view the reply, including the Loopback netmask.

Execute the RESTCONF GET Loopback API



The screenshot shows the RESTCONF GET Loopback API interface. The URL is `/data/Cisco-IOS-XE-native:native/interface/Loopback`. The interface includes a "Parameters" section with "No parameters" and a "Cancel" button. Below is a "Responses" section with a large blue "Execute" button.

16. Find the response within YANG Suite. If the request is successful, a response of 200 will be returned with the JSON response body. If the response body doesn't contain any information, a response of 204 will be returned, indicating a successful request with no data for that request found.

View, Copy, or Download the Response

The screenshot displays the 'Responses' section of the YANG Suite. It shows a curl command used to request data from a Cisco IOS-XE device. The request URL is `http://localhost:18480/restconf/proxy/https://10.1.1.5:443/restconf/data/Cisco-IOS-XE-native:interface/Loopback`. The server response is a 200 status code with a JSON body. The JSON body contains information about the Loopback interface, including its name, IP address (192.168.12.1), and mask (255.255.0.0). The response headers indicate that the content is private and should not be cached.

```
curl -X 'GET' \
'http://localhost:18480/restconf/proxy/https://10.1.1.5:443/restconf/data/Cisco-IOS-XE-native:interface/Loopback' \
-H 'accept: application/yang-data+json'
```

Request URL

```
http://localhost:18480/restconf/proxy/https://10.1.1.5:443/restconf/data/Cisco-IOS-XE-native:interface/Loopback
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "Cisco-IOS-XE-native:Loopback": [{ "name": 0, "ip": { "address": { "primary": { "address": "192.168.12.1", "mask": "255.255.0" } } }, "logging": { "event": { "link-status": [null] } } }] }</pre> <p>Response headers</p> <pre>cache-control: private,no-cache,must-revalidate,proxy-revalidate content-length: 325</pre>

Close

YANG Suite: gNMI

The gNMI tooling can be used to retrieve information from the device. With the help of programmatic interfaces and YANG Suite, it has become easier to retrieve operational information from Cisco IOS XE devices. YANG Suite provides a YANG API testing and validation environment that supports gNMI. The gNMI tooling uses JSON_IETF to encode data in its content layer. In this example, we run RPC for one of the wireless modules, Cisco-IOS-XE-wireless-access-point-oper.

Steps to access the gNMI plugin using YANG Suite

1. From the left navigation pane, select Protocols > gNMI.
2. Select the created YANG set.
3. Select the Module(s): left-interfaces.
4. Click "Load Modules."
5. From the drop-down list, select the device.
6. For Origin, select RFC7951.
7. In the Nodes section, make sure to expand the loaded YANG data model. Click the Value column next to the interfaces row. To narrow down the search, each individual node can be selected from the list.

- To generate the payload, click the blue “Build RPC” button.
- Once the payload is generated, it is ready to run by clicking “Run RPC(s).”

The screenshot shows the Cisco YANG Suite interface for gNMI GET Interfaces. The main window displays a tree view of YANG nodes. The 'interface' node is selected and expanded, showing its sub-nodes. The 'Build RPC' button is highlighted in blue. To the right, the generated RPC payload is displayed in JSON format:

```

{
  "path": [
    {
      "origin": "rfc7951",
      "elem": [
        {
          "name": "ietf-interfaces:interfaces"
        }
      ]
    }
  ],
  "encoding": "JSON_IETF"
}

```

The gNMI GET, gNMI GET Response, and gNMI Response values decoded will show up in a separate browser window.

The screenshot shows a browser window titled "gNMI GET Interfaces Response". The window displays the raw gRPC GET and GET Response messages, along with the decoded JSON response.

gRPC GET

```

path {
  origin: "rfc7951"
  elem {
    name: "ietf-interfaces:interfaces"
  }
}
encoding: JSON_IETF

```

gRPC GET Response

```

notification {
  timestamp: 16818491396916120
  update {
    path {
      origin: "rfc7951"
      elem {
        name: "ietf-interfaces:interfaces"
      }
    }
    val {
      json_ietf_val: "[{\"interface\": [{\"name\": \"PortyGigabitEthernet1/1/1\", \"type\": \"iana-if-type:ethernetCeraad\", \"enabled\": true, \"ietf-ipv4\": {}, \"ietf-ipv6\": {}}], [\"interface\": [{\"name\": \"PortyGigabitEthernet1/1/2\", \"type\": \"iana-if-type:ethernetCeraad\", \"enabled\": true, \"ietf-ipv4\": {}, \"ietf-ipv6\": {}}]]"
    }
  }
}

```

JSON Decoded

```

{
  "interfaces": [
    {
      "name": "PortyGigabitEthernet1/1/1",
      "type": "iana-if-type:ethernetCeraad",
      "enabled": true,
      "ietf-ipv4": {},
      "ietf-ipv6": {}
    },
    {
      "name": "PortyGigabitEthernet1/1/2",
      "type": "iana-if-type:ethernetCeraad",
      "enabled": true,
      "ietf-ipv4": {},
      "ietf-ipv6": {}
    }
  ]
}

```


Tooling: Cisco pyATS

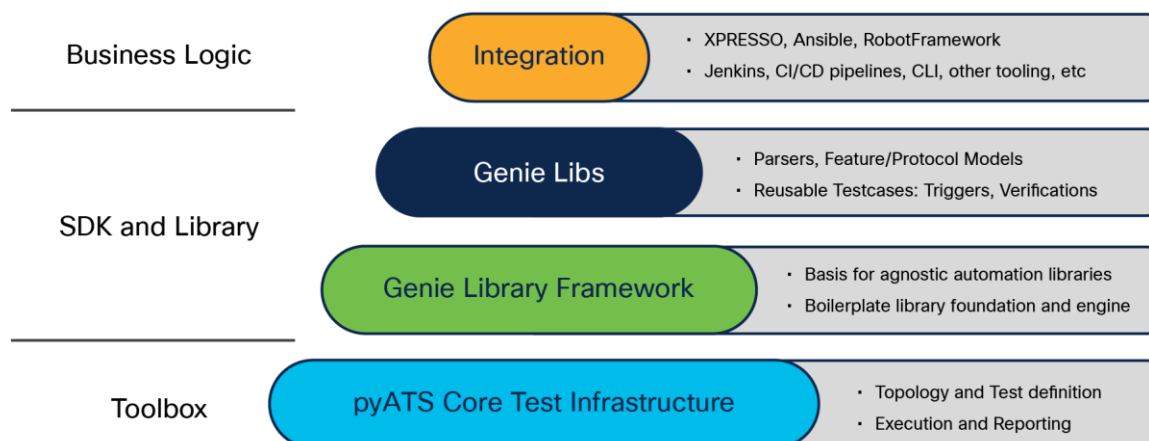


Figure 19.
Cisco pyATS structure

pyATS (pronounced “pie” and the individual letters “A,” “T,” “S”) is an end-to-end DevOps automation ecosystem publicly introduced in 2017. pyATS is agnostic by design, and it enables network engineers to automate their day-to-day DevOps activities, perform stateful validation of their device operational status, build a safety net of scalable, data-driven, reusable tests around their network, and visualize everything in a modern, easy-to-use dashboard. pyATS has a long history within Cisco, where it has been used as a consolidated test harness to run hundreds of thousands of tests against a variety of hardware and software features. It supports thousands of parsers and is customizable in the sense that new parsers can be submitted in addition to the currently available parsers. It is a consistent test harness for multiplatform and multivendor test activities and can be used with any of the programmatic interfaces and more.[]

pyATS has integrations for Cisco IOS XE’s programmatic interfaces as well as the legacy SSH CLI, which enables a very powerful combination of tooling options. pyATS works with the CLI and REST APIs using the REST connector. One common misconception about pyATS is that it is limited to just testing; actually, it also supports device configuration. Blitz is the pyATS YAML-abstracted implementation, which does not require Python.

Python Automation Test System

pyATS
print (network.profile)

pyATS provides sanity, feature, solution, system, and scale test & verification automation for products ranging from routers and switches, to access points, firewalls and cable CPEs.

It allows the device connections via CLI, NETCONF, or RESTCONF.

```
extends: base_tb_config.yaml

testbed:
  name: sampleTestbed
  alias: topologySampleTestbed
  credentials:
    default:
      username: admin
      password: CSC012345*
    enable:
      password: "MASK(user specified prompt)"
  servers:
    filesvr:
      server: ott2lab-tftp1
      address: 223.255.254.254
      path: ""
      credentials:
        default:
          username: rcuser
          password: 123rcpl
        sftp:
          username: sftpuser
          password: "M1NC(w6D0ms0Lw6fDqs00w5b0IQ==)"
          ftp:
            username: ftpuser
            password: "MASK(*)"
    http:
      server: 102.0.0.102
  custom:
    owner: john
    contacts: mail@domain.com
    mobile: "MASK(enter owner mobile phone number)"
```

```
devices:
  ott-101-n7x6:
    nci: nci66
    type: Nexus 7000
    alias: device-1
    credentials:
      default:
        username: admin
        password: abc123
      enable:
        password: "MASK(*)"
    connections:
      a:
        protocol: telnet
        ip: 10.85.84.80
        port: 2001
      b:
        protocol: telnet
        ip: 10.85.84.80
        port: 2003
      vty:
        protocol: telnet
        ip: 5.19.27.5
        credentials:
          default:
            username: ng1user
            password: ng1pw
    cclean:
      pre_clean: |
        switchname %(self)
        license grace-period
        feature telnet
        interface ng1a0
        ip addr %(self.connections.vty.ip)/24
        no shut
        vrf context management
        ip route 101.0.0.0/24 5.19.27.251
        ip route 102.0.0.0/24 5.19.27.251
      post_clean: |
        switchname %(self)
        license grace-period
        feature telnet
        interface ng1a0
        ip addr %(self.connections.vty.ip)/24
        no shut
        vrf context management
        ip route 101.0.0.0/24 5.19.27.251
        ip route 102.0.0.0/24 5.19.27.251
    custom:
      SWP1: Supervisor Module-1X
      SWP2: Supervisor Module-1X
```

<https://developer.cisco.com/pyats/>
<https://developer.cisco.com/docs/pyats/api/>

Figure 20.
Benefits of Cisco pyATS

Learn more about pyATS:

- Explore pyATS: <https://developer.cisco.com/pyats>
- Read the docs: <https://developer.cisco.com/docs/pyats/api/>

Tooling: Cisco Network Services Orchestrator (NSO)

The Cisco NSO is another powerful controller solution that abstracts network intent and maintains configuration state. Additional details are available from developer.cisco.com/site/nso/.

Tooling: Ansible

Ansible is a popular and easy-to-use open-source software suite that automates software provisioning, configuration, and management. It connects to and controls devices via SSH, NETCONF, and a variety of other protocols as well. Ansible is agentless, meaning there is no installation and no requirements on the target device, other than having an accessible API or interface. It is minimal in nature and provides a secure and reliable way to interact with remote devices. Ansible is highly adaptable and commonly used with other automation tools to accomplish complex workflows. Below are some examples of using Ansible to complete basic day 0 configuration tasks.

Ansible Taxonomy

- **Role:** a set of Playbooks ()
- **Playbook:** repeatable standard config
- **Play:** a set of tasks
- **Task:** single action that references a module
- **Module:** reusable, standalone scripts

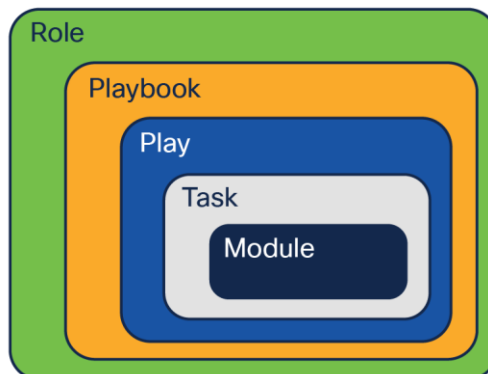


Figure 21.

Ansible taxonomy

Ansible has several components that work together to provide a holistic solution. Modules are reusable, standalone scripts. Tasks call upon modules to perform an action. When there are multiple tasks, a play can be used to call the tasks in a particular order. A playbook is then used when there are multiple plays. Finally, a role is a set of playbooks.

CLI example to enable APIs

The hosts file has connection details and device-specific information, including credentials. In this example, it is assumed that the device has already been configured to allow SSH logins, and the enable password has been set. The hosts files and YAML file are used together to accomplish a task. In this case, the tasks are to connect to the CLI over SSH, enter enable mode, and execute the required Cisco IOS commands to enable netconf-yang and set up the AAA requirements.

Ansible hosts file for day 0 configuration

Create the default config file ansible.cfg

```
#ansible.cfg
[defaults]
inventory = ./host
host_key_checking = False
remote_user = admin
```

This example hosts file contains the variables needed to successfully establish a connection to the device.

Note: In the example below, replace “admin” with the device’s SSH username and “<DEVICE_PASSWORD>” with the device’s SSH password.

host

```
# hosts
[C9300]
10.1.1.5

[all:vars]
ansible_connection=network_cli
ansible_network_os=ios
ansible_ssh_user=admin
ansible_ssh_pass=<DEVICE PASSWORD>
```

Ansible YAML configuration file to enable NETCONF and RESTCONF

The following example YAML file enable_netconf_yang.yaml can be used to enable the NETCONF interface on the device and configure the authentication prerequisites, including adding a user.

```
- hosts: C9300
  gather_facts: no

  tasks:
    - ios_config:
        commands:
          - aaa new-model
          - aaa authorization exec default local
          - aaa authentication login default local
          - username netconf privilege 15 password 0 netconf
          - netconf-yang
          - ip http secure-server
          - restconf
        save_when: modified
```

CLI example to show API status

This example `cat9300_verify.yaml` YAML file will run two Cisco IOS XE show commands to verify that `netconf-yang` and `restconf` are enabled, and the output will be registered and displayed on the screen when executed.

```
- hosts: C9300
  gather_facts: no

  tasks:
    - ios_command:
        commands:
          - show run | i netconf-yang
          - show run | i restconf
        register: show
    - debug: var=show.stdout_lines
```

Executing the task to enable and verify NETCONF and RESTCONF

The `ansible-playbook` command can be used to execute the task that we define above to enable the NETCONF-YANG interface on the device. In this example, we define a variable to set Host Key Checking to false, so that the SSH host key is not validated. In production environments, it is important to verify the authenticity of the device being accessed. However, in this example, the check is set to false for ease of use.

From the command line, execute the following command to run the `enable_netconf_yang.yaml` configuration:

```
$ ansible-playbook enable_netconf_yang.yaml
```



```
auto@pod21-xelab:~/ansible/ansible-intro/day0$ ansible-playbook enable_netconf_yang.yaml
PLAY [c9300] *****
TASK [ios_config] *****
[DEPRECATION WARNING]: Distribution ubuntu 20.04 on host 10.1.1.5 should use /usr/bin/python3, but is using /usr/bin/python for backward compatibility with prior Ansible releases. A future Ansible release will default to using the discovered platform python for this host. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more information. This feature will be removed in version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
changed: [10.1.1.5]
PLAY RECAP *****
10.1.1.5      : ok=1  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
auto@pod21-xelab:~/ansible/ansible-intro/day0$
```

From the command line, execute the following command to run the `cat9300_verify.yaml` configuration:

```
$ ansible-playbook cat9300_verify.yaml
```

```

auto@pod21-xelab:~/ansible/ansible-intro/day0$ ansible-playbook cat9300_verify.yaml
PLAY [c9300] *****
TASK [ios_command] *****
[DEPRECATION WARNING]: Distribution ubuntu 20.04 on host 10.1.1.5 should use /usr/bin/python3, but is using /usr/bin/python for backward compatibility with
prior Ansible releases. A future Ansible release will default to using the discovered platform python for this host. See
https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more information. This feature will be removed in version 2.12.
Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
ok: [10.1.1.5]
TASK [debug] *****
ok: [10.1.1.5] => {
  "show.stdout_lines": [
    [
      "netconf-yang",
      "netconf-yang ssh local-vrf guestshell enable"
    ],
    [
      "restconf"
    ]
  ]
}
PLAY RECAP *****
10.1.1.5 : ok=2  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
auto@pod21-xelab:~/ansible/ansible-intro/day0$

```

NETCONF XML example

Create a subscription to CPU utilization on our device using Ansible plus NETCONF. This example sends XML into the NETCONF interface to add the subscription. Create three files in a “day1” folder: host, ansible.cfg, and add_sub.yaml. The content for each file is listed below. Make sure to change the IP address, username, and password.

Modify the host file from the example above to the following:

```

[c9300]
10.1.1.5

[all:vars]
ansible_connection=netconf
ansible_network_os=ios
ansible_password=<DEVICE_PASSWORD>

ansible.cfg
[defaults]
inventory = ./host
host_key_checking = False
remote_user = <USERNAME>

```

add_sub.yaml

```
---
- hosts: c9300
  gather_facts: no
  connection: netconf
  remote_user: admin

  tasks:
  - name: establish subscription
    netconf_config:
      xml: |
        <nc:config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <mdt-config-data xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-mdt-cfg">
            <mdt-subscription>
              <subscription-id>501</subscription-id>
              <base>
                <stream>yang-push</stream>
                <encoding>encode-kvgpb</encoding>
                <source-address>10.60.0.19</source-address>
                <source-vrf>Mgmt-vrf</source-vrf>
                <period>2000</period>
                <xpath>/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-
seconds</xpath>
              </base>
              <mdt-receivers>
                <address>10.12.252.224</address>
                <port>57500</port>
                <protocol>grpc-tcp</protocol>
              </mdt-receivers>
            </mdt-subscription>
          </mdt-config-data>
        </nc:config>
```

From the command line, execute the following command to run the add_sub.yaml configuration:

```
$ ansible-playbook add_sub.yaml
```

```

auto@pod21-xelab:~/ansible/ansible-intro/day1$ ansible-playbook add_sub.yaml
PLAY [c9300] *****
TASK [establish subscription] *****
unable to load netconf plugin for network_os ios, falling back to default plugin
[WARNING]: Platform linux on host 10.1.1.5 is using the discovered Python interpreter at /usr/bin/python, but future installation of another Python interpreter
could change this. See https://docs.ansible.com/ansible/2.9/reference_appendices/interpreter_discovery.html for more information.
ok: [10.1.1.5]
PLAY RECAP *****
10.1.1.5 : ok=1  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
auto@pod21-xelab:~/ansible/ansible-intro/day1$

```

Additional examples and resources for Ansible can be found at <https://github.com/jeremycohoe/ansible-config-samples>.

RESTCONF JSON example

Ansible also has a rich history of integration with REST, which includes the RESTCONF interface. YANG Suite can be used to work with the RESTCONF interface and has optional features to export an Ansible playbook based on the YANG payload that was built within YANG Suite.

The screenshot shows the YANG Suite RESTCONF interface. At the top, there are dropdown menus for 'Select a YANG set' (c9300-default-yangset) and 'Select a device' (C9300). To the right, there are fields for 'Select YANG module(s)' (Cisco-IOS-XE-interfaces-oper) and 'Select depth limit' (No limit). Below these are buttons for 'Load module(s)', 'Generate API(s)', 'Show API(s)', 'Generate Python Script', and 'Download Ansible Playbook'. A tree view shows the 'Cisco-IOS-XE-interfaces-oper' module expanded to 'interfaces', then 'interface', and finally a list of fields: name, interface-type, admin-status, oper-status, last-change, if-index, and phys-address. Below the tree, the text 'Download Ansible Playbook' is displayed. Underneath, there is a section 'Fill Out All Fields' with a 'Selected XPath' field containing '/interfaces'. At the bottom, there are fields for 'Script File Name' (restconf.yaml), 'Ansible Task Name' (Task 1), 'REST Message Name' (REST), and 'Select a REST Method' (GET).

A YAML payload is generated that can be run directly within Ansible after ensuring that the hosts, inventory, and configuration defaults are set accordingly:

```
# Example of basic ansible_host file referred to in
# ansible.cfg inventory:
#
#[HOST_NAME_HERE]
#IP_ADDRESS_HERE
#
#[HOST_NAME_HERE:vars]
# ansible_connection: httpapi
# ansible_network_os: restconf
# ansible_httpapi_use_ssl: true
# ansible_httpapi_validate_certs: false
# ansible_httpapi_port: 443
# ansible_httpapi_restconf_root: /restconf/data/
# ansible_user: USERNAME_HERE
# ansible_password: PASSWORD_HERE
#

- name: REST
  hosts: HOST_NAME_HERE
  gather_facts: no
  tasks:
    - name: Task 1
      ansible.netcommon.restconf_get:
        # Output can either be json or xml
        output: json
        path: Cisco-IOS-XE-interfaces-oper:interfaces
```

Tooling: Terraform

Terraform is an [Infrastructure-as-Code](#) (IaC) tooling that allows network operators to easily view operational data, configure devices, and manage network resources. Since Terraform is cloud native, it works well with Cisco IOS XE cloud-native solutions for routing, switching, and wireless platforms, including the Cisco Catalyst 9000 switch family, the Cisco Catalyst 8000V (virtual) router, and the Cisco Catalyst 9800-CL Wireless Controller for Cloud. In addition to easily managing cloud-native solutions, Terraform can configure campus solutions. With Cisco IOS XE, we can automate with any tooling on any interface.

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

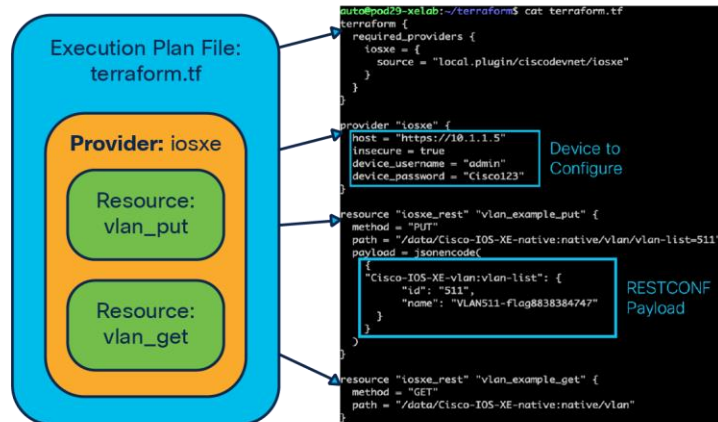


Figure 22.
Terraform terminology

Getting started with the Terraform Cisco IOS XE provider

1. Install Terraform: <https://learn.hashicorp.com/tutorials/terraform/install-cli>.
2. Clone the Cisco IOS XE Terraform provider: <https://github.com/CiscoDevNet/terraform-provider-iosxe/>.
3. Create a .tf file.
4. Run Terraform using “terraform init; terraform apply.”

Add an ACL to a Catalyst 9300X using Terraform

Create a Terraform file using the [CiscoDevNet/iosxe Terraform provider](#) as in the following example.

add_acl.tf

```
terraform {
  required_providers {
    iosxe = {
      version = "0.1.1"
      source  = "CiscoDevNet/iosxe"
    }
  }
}

provider "iosxe" {
  request_timeout = 30
  insecure       = true
}

# Adding extended ACL
resource "iosxe_rest" "acl_example_post" {
  method = "POST"
  path   = "/data/Cisco-IOS-XE-native:native/ip/access-list"
  payload = jsonencode(
    {
      "Cisco-IOS-XE-acl:extended": [
        {
          "name": 102,
          "access-list-seq-rule": [
            {
              "sequence": "10",
              "ace-rule": {
                "action": "permit",
                "protocol": "ip",
                "host-address": "10.1.1.3",
                "dst-any": [
                  null
                ],
                "precedence": "routine",
                "tos": "normal",
                "log": [
                  null
                ]
              }
            }
          ]
        }
      ]
    }
  )
}
```

```
    ]
  }
},
{
  "sequence": "20",
  "ace-rule": {
    "action": "permit",
    "protocol": "tcp",
    "any": [
      null
    ],
    "dst-any": [
      null
    ],
    "dst-eq": 600
  }
},
{
  "sequence": "30",
  "ace-rule": {
    "action": "permit",
    "protocol": "udp",
    "any": [
      null
    ],
    "dst-any": [
      null
    ],
    "dst-eq": 200
  }
},
{
  "sequence": "40",
  "ace-rule": {
    "action": "permit",
    "protocol": "icmp",
    "any": [
      null
    ],
    "dst-any": [
      null
    ],
  ],
```

```

        "dst-eq-port2": 250
      }
    },
    {
      "sequence": "50",
      "ace-rule": {
        "action": "permit",
        "protocol": "igmp",
        "any": [
          null
        ],
        "dst-any": [
          null
        ],
        "dst-eq-port2": 15
      }
    }
  ]
}
)
}

```

Next, run the Terraform file using the following commands:

```

terraform init
terraform apply -auto-approve

```

The device is now configured with an ACL as specified in the add_acl.tf file.

Learn more about Terraform

- GitHub provider examples: <https://github.com/CiscoDevNet/terraform-provider-iosxe/>
- Provider binary: <https://registry.terraform.io/search/providers?namespace=CiscoDevNet>
- Go client: <https://github.com/CiscoDevNet/iosxe-go-client>
- Blogs: <https://blogs.cisco.com/tag/terraform>
- Intro to Terraform video: https://www.youtube.com/watch?v=GEY_hyXimbA
- Configure an IPsec tunnel with Terraform: <https://www.youtube.com/watch?v=bPS0bhPacDw>
- Now that the go client has been released, individuals are contributing their own Terraform providers, such as this BGP EVPN provider: <https://github.com/robertcsapo/terraform-provider-ciscoevpn>

Day 2: Model-driven telemetry

The ietf-yang-push model is used for NETCONF streaming telemetry. In the following example, a NETCONF subscription is created using NETCONF in YANG Suite. To create this subscription, the stream is set to “ys:yang-push,” the encoding is “notif-bis:encode-xml,” the filter is a specific xpath “/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds,” and yp:periodic updates every 1000 centiseconds (10 seconds). Once each of the leaves is created in YANG Suite, click the blue “Build RPC” button and then click the blue “Run RPC(s)” button.

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <establish-subscription xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications">
    <stream xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">yp:yang-push</stream>
    <encoding>encode-xml</encoding>
    <xpath-filter xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds</xpath-filter>
    <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">1000</period>
  </establish-subscription>
</rpc>
```

NETCONF and SNMP event streams

Two main event streams are used for model-driven telemetry, “yang-push” and “yang-notif-native,” which handle publication of the periodic as well as the event-based telemetry. In addition to the NETCONF streams, there is also a stream from the SNMP events. This stream is used when collecting SNMP traps or events over the NETCONF interface.

Event Streams

The screenshot shows the YANG Suite NETCONF interface. At the top, there's a header with the YANG Suite logo and 'NETCONF'. Below that, there are several input fields and buttons: 'YANG Set' (dropdown), 'Module(s)' (text input with placeholder 'Type here to filter available modules'), 'NETCONF Operation' (dropdown set to 'get-config'), 'Device' (dropdown set to 'c9300'), 'Edit Device' (gear icon button), and 'Open Device Window' (dropdown). Below these are buttons for 'YANG Tree', 'Replays', 'RPC Options...', 'Build RPC', and 'Run RPC(s)'. The 'Actions:' menu is open, showing a list of actions: 'Report device capabilities', 'Event notifications (RFC 5277)', 'get streams' (highlighted in blue), 'subscribe', 'Candidate datastore', 'get-config', 'validate', 'commit', 'discard', 'Running datastore', 'get-config', 'Startup datastore', and 'get-config'. The right pane displays the XML output of the 'get streams' action:

```
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:8869c95c-1bd3-4a6b-b281-d7f1e269c03c"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:pa
<data>
  <netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">
    <streams>
      <stream>
        <name>NETCONF</name>
        <description>default NETCONF event stream</description>
        <replySupport>false</replySupport>
      </stream>
      <stream>
        <name>snmpevents</name>
        <description>SNMP related notifications</description>
        <replySupport>true</replySupport>
        <replyLogCreationTime>2022-01-13T15:07:28+00:00</replyLog
      </stream>
    </streams>
  </netconf>
</data>
</rpc-reply>
```

SNMP and screen scraping from the CLI have been used as traditional network monitoring interfaces for years. Since the technology is widely popular in network management for network monitoring, network engineers still actively use it. The SNMP messages are transported via User Datagram Protocol (UDP) and always require active polling by the collector, which is not always reliable. The read-write mode can make a network vulnerable to attacks. Other limitations include the security concerns of SNMP, lack of information about the source IP address, what type of traffic is sent, and information about the destination IP address. In other words, the data is mainly unstructured, and the format frequently changes between software releases. Nevertheless, more recent versions of SNMP bring improvements in security, performance, and flexibility.

In contrast to SNMP, NetFlow was designed for network monitoring. It brings more visibility into the network and explicitly gives information about the source IP address, application protocol, and destination IP address, which SNMP lacks. NetFlow works in the same way as SNMP: It sends records from a cache to a collector, and all the records are being pushed to the collector without being requested each time. However, NetFlow is not used for collecting information about bandwidth, CPU utilization, memory, and/or the temperature of the device.

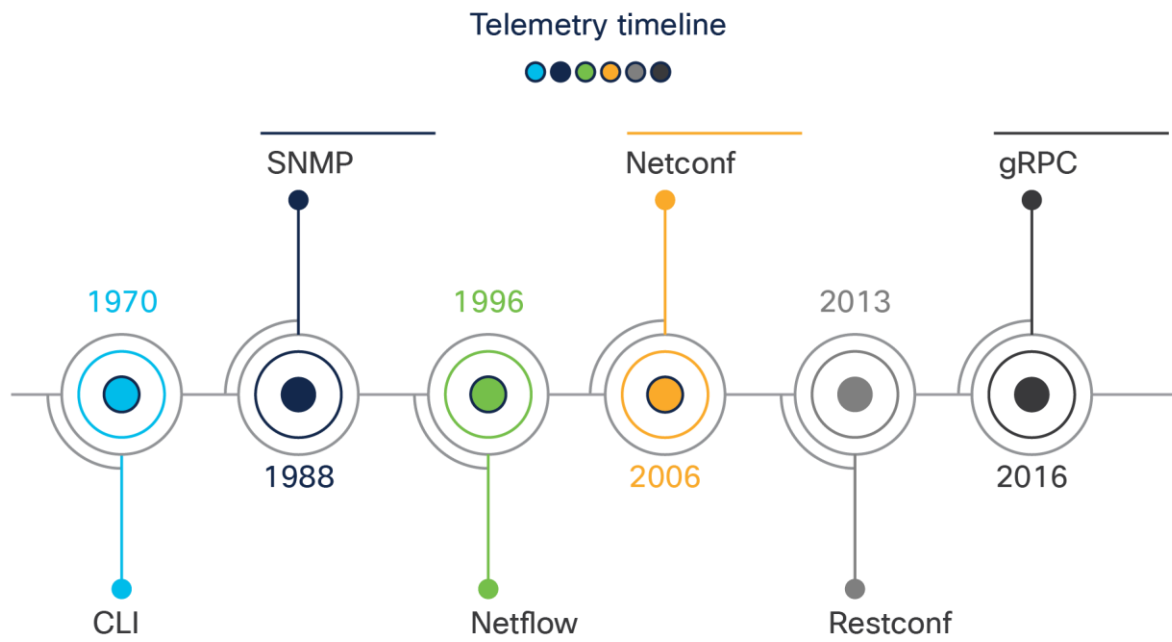


Figure 23.
Telemetry standards timeline

Over the past couple of years, we have adopted new technologies that have helped us solve some, but not all of the predecessors' flaws in the modern world. Cisco IOS XE supports the YANG data modeling language, which can be used with NETCONF to deliver the desired programmable and automated network operations. NETCONF is an XML-based protocol used to install, manipulate, and delete the configuration of network devices via SSH as the transport layer. It is based on an RPC mechanism to provide communication between a client and a server. In our case, the server is the network device and the controller is the client.

The web is progressing at an exponential speed, and we've started seeing quick adoption of RESTCONF in the web-based monitoring stack. RESTCONF uses HTTP methods to implement similar NETCONF operations for accessing data defined in YANG. In comparison to NETCONF, RESTCONF supports both XML and JSON encodings. But it's important to clarify that RESTCONF is not a NETCONF replacement and was never intended to be one. From a capabilities standpoint, RESTCONF has its limitations, like any other network automation tool. It lacks any type of validation and also lacks the "lock" concept found in NETCONF.

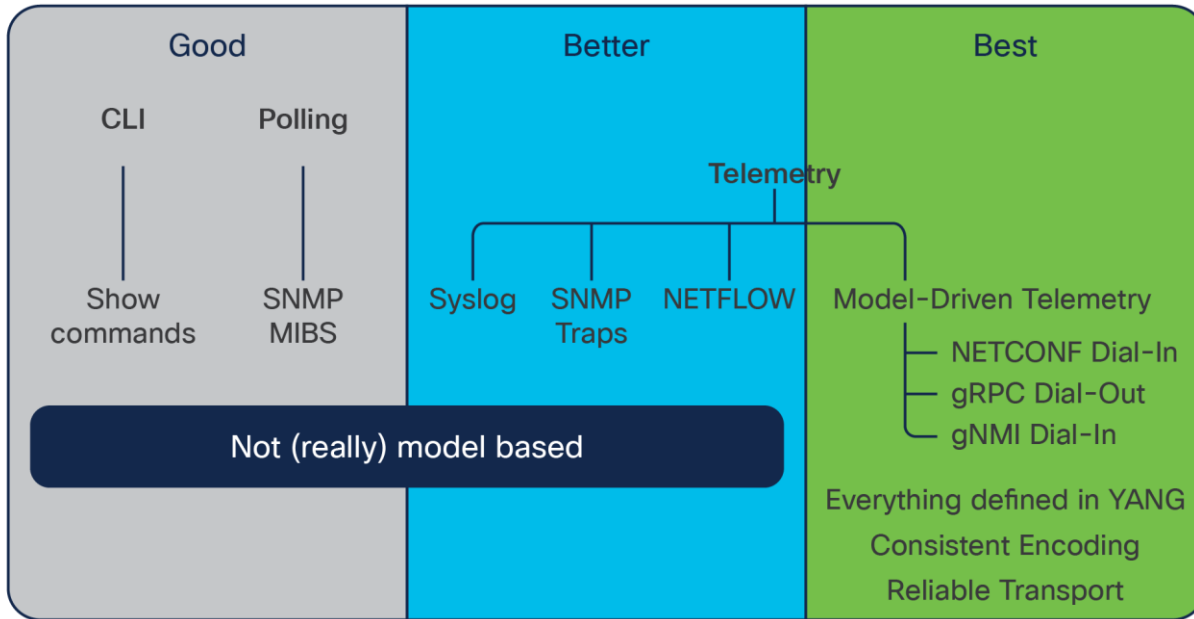


Figure 24.
Comparison chart: CLI, polling, and telemetry monitoring

In addition, we have gRPC, a modern open-source RPC using HTTP for APIs. Model-driven telemetry with gRPC addresses many of the shortfalls of the legacy monitoring capabilities and provides an additional interface from which telemetry is now available to be published. gRPC is a YANG model using JSON and protobuf encodings. Unlike the NETCONF telemetry interface, which is “dial-in” and session-based, the gRPC interface is “dial out” and based on configuration within the device. gRPC is push-based, meaning that once it is configured, it will send the requested telemetry data regularly to the provided recipient(s) without them needing to request the data. Now you can decide what data is needed, how often, and where to send it. Once the configuration is in place, the Cisco IOS XE device easily publishes the telemetry data to third-party collectors, your monitoring tools, extensive data search and visualization engines such as Splunk and Elastic, or even a simple text file.

Finally, while the CLI and SNMP aren’t going away anytime soon, automation is a big part of where networks are headed. Protocols such as YANG, NETCONF, RESTCONF, and gRPC were designed with this in mind. That's why Cisco uses them within its platforms.

NETCONF dial-in model-driven telemetry

NETCONF can also be used as a dial-in model-driven telemetry interface that uses dynamic telemetry subscriptions.

Dynamic telemetry subscription with Python NCC

Dial-in telemetry subscriptions can be easily created by using the NCC tools available from CiscoDevNet at <https://github.com/CiscoDevNet/ncc>. The repository can simply be cloned from GitHub with the “git clone” command. An additional requirement is to use a patched ncclient tool that works with the ncc-establish-subscription.py tool. This can be installed by following the directions on the ncc GitHub page or by executing the two commands below:

```
git clone https://github.com/CiscoDevNet/ncc
sudo pip install --upgrade git+https://github.com/CiscoDevNet/ncclient.git
```

Once downloaded, a subscription can be established by running the following command:

```
python3 ncc-establish-subscription.py --host 10.1.1.5 --port 830 -u admin -p Cisco123 --
period 1000 -x "/ios:native"
```

```

(v) auto@pod21-xelab:~/ncc$
(v) auto@pod21-xelab:~/ncc$ python3 ncc-establish-subscription.py --host 10.1.1.5
--port 830 -u admin -p Cisco123 --period 1000 -x "/ios:native"
/home/auto/ncc/v/lib/python3.8/site-packages/paramiko/kex_ecdh_nist.py:39: Cryptogr
aphyDeprecationWarning: encode_point has been deprecated on EllipticCurvePublicNumb
ers and will be removed in a future version. Please use EllipticCurvePublicKey.publ
ic_bytes to obtain both compressed and uncompressed point encoding.
  m.add_string(self.Q_C.public_numbers().encode_point())
/home/auto/ncc/v/lib/python3.8/site-packages/paramiko/kex_ecdh_nist.py:91: Cryptogr
aphyDeprecationWarning: Support for unsafe construction of public numbers from enco
ded data will be removed in a future version. Please use EllipticCurvePublicKey.fro
m_encoded_point
  self.Q_S = ec.EllipticCurvePublicNumbers.from_encoded_point(
/home/auto/ncc/v/lib/python3.8/site-packages/paramiko/kex_ecdh_nist.py:103: Cryptog
raphyDeprecationWarning: encode_point has been deprecated on EllipticCurvePublicNum
bers and will be removed in a future version. Please use EllipticCurvePublicKey.pub
lic_bytes to obtain both compressed and uncompressed point encoding.
  hm.add_string(self.Q_C.public_numbers().encode_point())
Subscription Result : notif-bis:ok
Subscription Id      : 2147483653
-->>
(Default Callback)
Event time          : 2022-06-05 23:40:33.330000+00:00
Subscription Id    : 2147483653
Type               : 1
Data               :
<datastore-contents-xml xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <version>17.8</version>
    <memory>
      <free>
        <low-watermark>
          <processor>131046</processor>
        </low-watermark>

```

In the example above, a telemetry subscription has been created using the Cisco-IOS-XE-native YANG model, which has a prefix of “IOS” and an xpath of “native.” YANG Suite is used to determine the correct xpath filter for this YANG model, as seen in the screenshot below:

YANG Suite / Exploring YANG / YANG set "c9300-default-yangset" / Modules

admin

Explore YANG Models

Select a YANG set: c9300-default-yangset

Select YANG module(s): Cisco-IOS-XE-native x

Load module(s)

Icon legend Search XPath(s) Search nodes Expand all nodes

Display schema nodes only
 Display all nodes

Cisco-IOS-XE-native

- native
 - default
 - bfd
 - version
 - stackwise-virtual
 - boot-start-marker
 - boot
 - boot-end-marker
 - banner
 - captive-portal-bypass
 - memory
 - location
 - call-home

Node Properties

Name	native
Nodetype	container
Description	
Module	Cisco-IOS-XE-native
Revision	2022-03-01
Xpath	/native
Prefix	ios
Namespace	http://cisco.com/ns/yang/Cisco-IOS-XE-native
Schema Node Id	/native
Access	read-write
Operations	<ul style="list-style-type: none"> "edit-config" "get-config" "get"

gNMI dial-in model-driven telemetry

gNMI is a gRPC network management interface. It provides the mechanism to install, manipulate, and delete the configuration of network devices, and to view operational data. The content provided through gNMI can be modeled using YANG. gRPC is an RPC developed by Google for low-latency, scalable distributions with mobile clients communicating to a cloud server. It carries gNMI and provides the means to formulate and transmit data and operation requests.

gRPC dial-out model-driven telemetry

gRPC is an RPC dial-out model-driven telemetry interface. gRPC dial-out telemetry is an automated communications process by which measurements and other data are collected and transmitted to the remote receiving equipment for monitoring. Model-driven telemetry provides a mechanism to stream YANG-modeled data to a data collector over the network.

Dial-in dynamic vs. dial-out configured MDT subscriptions

With a dial-in or “dynamic” subscription, the subscriber must first establish a session with a connection to the device and then subscribe to the data models. The NETCONF session must remain established for telemetry data to continue streaming. If the session is disconnected, the telemetry subscription must be manually reestablished.

With dial-out or “configured” subscriptions, once the configuration is set up by the user, the device will maintain the subscription configuration and send telemetry to the subscriber without needing an active session to the collector.

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

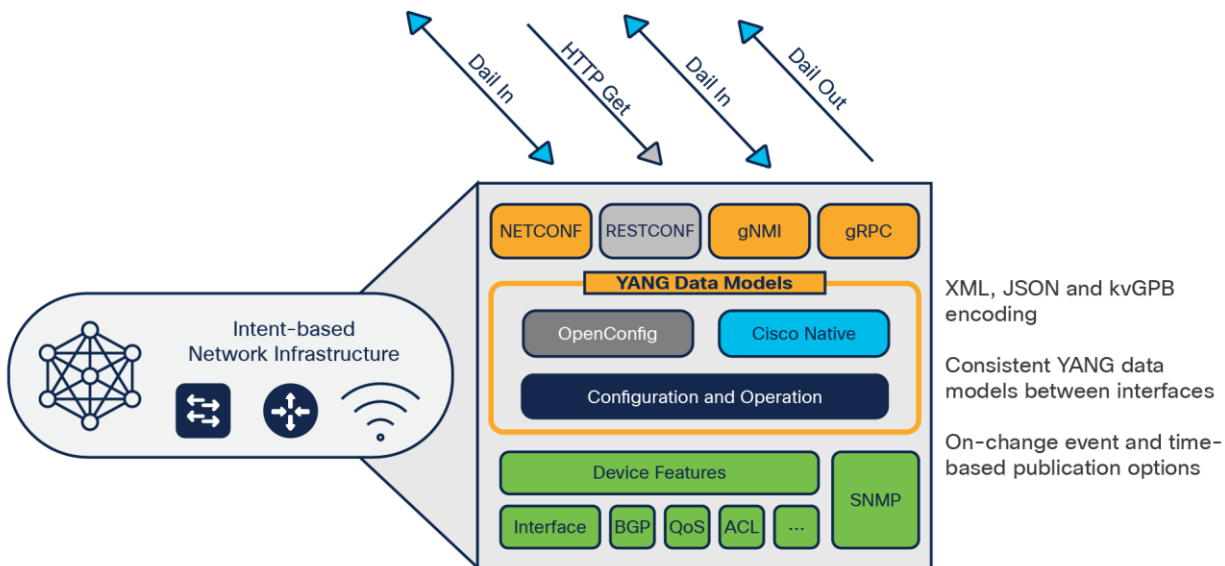


Figure 25.
Model-driven telemetry interfaces

gRPC dial-out model-driven telemetry configuration

gRPC dial-out subscriptions support encoding with key-value Google protocol buffers (kv-gpb) over a TCP connection. The following configuration can be used to establish a gRPC dial-out with FQDN DNS support telemetry subscription. Additionally, gRPC can be configured with TLS and mTLS to support high security and geographically dispersed collection use cases.

The following is an example of a dial-out configuration using the DNS named receiver to publish telemetry data about interface utilization every 10 seconds:

```
telemetry ietf subscription 101
  encoding encode-kvgpb
  filter xpath /interfaces-ios-xe-oper:interfaces/interface
  source-address 10.85.134.65
  stream yang-push
  update-policy periodic 1000
  receiver-type protocol
  receiver name yangsuite

telemetry receiver protocol yangsuite
  host name yangsuite-telemetry.cisco.com 57500
  protocol grpc-tcp
```

This configuration creates a new subscription with an ID of 101. The encoding is set to kv-gpb, and the xpath filter defines the API to subscribe to, in this case dot11-oper-data. The xpath filter is defined within the YANG model, and YANG Suite is used to determine the exact xpath and prefix for this model. The source address and Virtual Routing and Forwarding (VRF) to use from the device is set, as well as the receiver IP, port, and protocol. The yang-push stream defines how often to publish data in centiseconds. In this case it is set to 2000, which means data will be published every 20 seconds.

Receiving gRPC model-driven telemetry with Telegraf

The kv-gpb telemetry data that is sent over the gRPC interface can be received with many tools and in many different configurations, depending on the business needs and use cases. Telegraf is an open-source tool that can be used to receive the data and is available on GitHub at <https://github.com/influxdata/telegraf>. Telegraf works by acting as the gRPC server and receiver, where it processes the Google protocol buffers' encoded data and sends the text data into the time series database InfluxDB. From there, Grafana can be used to visualize the data. Telegraf and Grafana are highly configurable and can receive and visualize a variety of telemetry sources as well as output data to a variety of data sources, including Kafka, InfluxDB, Elasticsearch, and Prometheus.

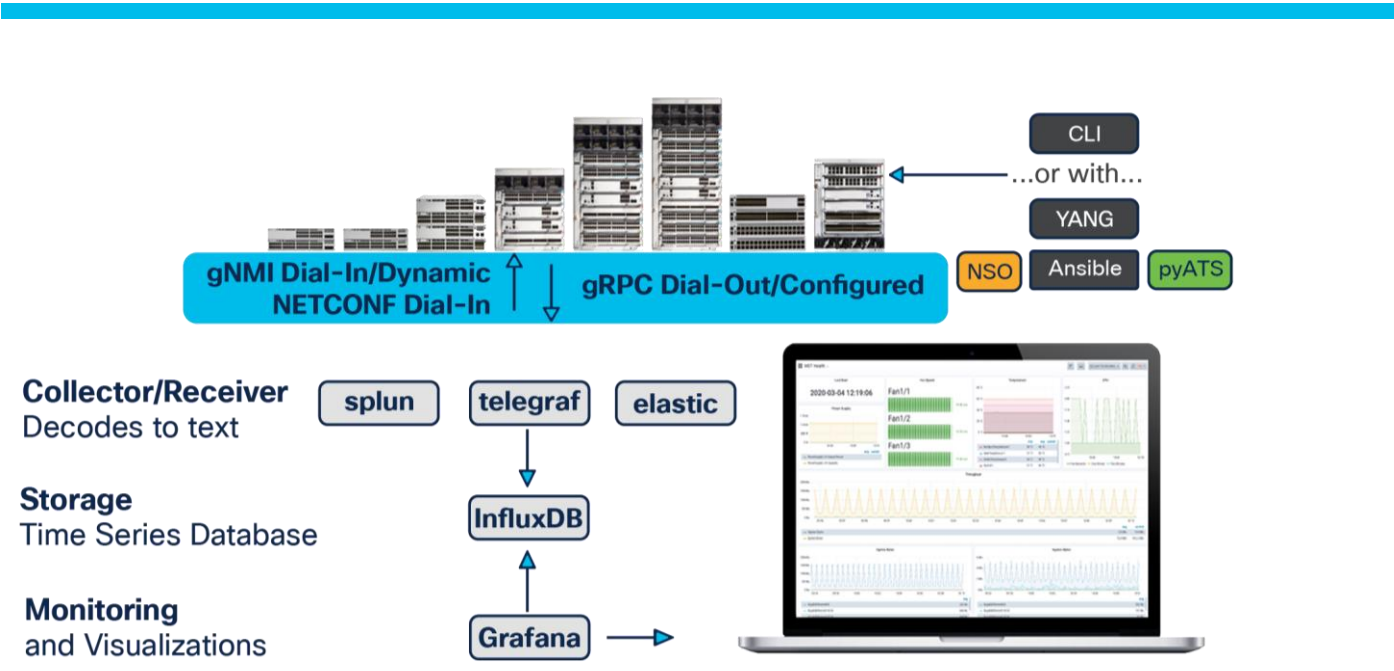


Figure 26.
gRPC workflow example

Verify telemetry subscriptions

Several show commands are available to verify the status of the telemetry subscription configurations.

Examples of each are below:

```
show telemetry ietf subscription all
```

```
c9300-pod21#show telemetry ietf subscription all
ID           Type        State      State Description
6041337      Configured  Valid      Subscription validated
2147483649   Dynamic     Valid      Subscription validated
2147483650   Dynamic     Valid      Subscription validated

c9300-pod21#
```

```
show telemetry ietf subscription <ID> detail
```



```
c9300-pod21#show telemetry ietf subscription 6041337 detail
Telemetry subscription detail:
```

```
Subscription ID: 6041337
Type: Configured
State: Valid
Stream: yang-push
Filter:
  Filter type: xpath
  XPath: /process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds
Update policy:
  Update Trigger: periodic
  Period: 30000
Encoding: encode-kvgpb
Source VRF:
Source Address:
Notes: Subscription validated
```

Named Receivers:

Name	Last State Change	State
------	-------------------	-------

grpc-tcp://10.1.1.3:57500	06/05/22 14:18:37	Connected

```
show telemetry ietf subscription <ID> receiver
```

```
c9300-pod21#show telemetry ietf subscription 6041337 receiver
Telemetry subscription receivers detail:
```

```
Subscription ID: 6041337
Name: grpc-tcp://10.1.1.3:57500
Connection: 3
State: Connected
Explanation:
```

Tooling: Cisco YANG Suite

YANG Suite can be used for streaming telemetry using NETCONF, gNMI, or gRPC.

NETCONF dial-in telemetry plugin

YANG Suite can be used to dial in to the NETCONF interface and to configure the dynamic telemetry subscription against the "ietf-event-notifications" YANG data model. Once the required fields and settings are set for the stream, encoding type, xpath, and update period, the XML payload can be sent to the device, which will start sending the requested telemetry data within the NETCONF session.

The screenshot shows the Cisco YANG Suite interface for a NETCONF session. The top bar indicates the session is for YANG Set 'C9300' and Module(s) 'Cisco-IOS-XE-interfaces-oper.x' and 'ietf-event-notifications.x'. The device is identified as 'C9300'. The interface includes buttons for 'Edit Device', 'Open Device Window', 'Build RPC', 'Run RPC(s)', and 'Clear RPC(s)'. The 'YANG Tree' on the left shows the configuration for the 'ietf-event-notifications' module, specifically the 'establish-subscription' node. The configuration table below the tree shows the following settings:

Nodes	Value
input	yp:yang-push
encoding	cyp:encode-kvq
filter-type	rfc5277
filter	
yp:update-filter	
yp:update-filter	
yp:subtree	
yp:xpath	
yp:xpath-filter	/process-cpu-los-x
cyp:native-filter	
by-reference	
yp:update-trigger	
yp:periodic	
yp:period	1000
yp:anchor-time	
yp:on-change	

The XML payload on the right is as follows:

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <establish-subscription xmlns="urn:ietf:params:xml:ns:yang:ietf-event-notifications">
    <stream xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">yp:yang-push</stream>
    <encoding xmlns:cyp="urn:cisco:params:xml:ns:yang:cisco-xe-ietf-yang-push-ext">cyp:encode-kvq</encoding>
    <xpath-filter xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">/process-cpu-los-x</xpath-filter>
    <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">1000</period>
  </establish-subscription>
</rpc>
```

gNMI dial-in telemetry plugin

Similar to NETCONF, the gNMI plugin to YANG Suite can be used to create and receive the telemetry data within the gNMI session.

The screenshot displays the gRPC Network Management Interface (gNMI) web application. The interface is titled "gRPC Network Management Interface (gNMI)" and shows the following configuration:

- YANG Set:** jcohoe-c9300-default-yangset
- Module(s):** Cisco-IOS-XE-interfaces-oper
- Device:** jcohoe-c9300
- Capabilities:** gNMI Operation (selected), Get, Set, Subscribe
- Origin:** RFC 7951 (selected), Openconfig, Other
- Encoding type:** JSON_IETF (selected), JSON
- Mode:** SAMPLE (selected), ON_CHANGE, POLL, ONCE, STREAM
- Sample interval:** 10
- Prefixing:** checked

The interface also features a tree view of nodes on the left, a "Build JSON" button, a "Clear Values" button, and "Run RPC(s)" and "Clear RPC(s)" buttons. The JSON RPC configuration is displayed in a text area on the right:

```
{
  "subscribe": {
    "prefix": {
      "origin": "rfc7951"
    },
    "subscription": [
      {
        "path": {
          "elem": [
            {
              "name": "Cisco-IOS-XE-interfaces-oper:interfaces"
            },
            {
              "name": "interface"
            }
          ]
        },
        "mode": "SAMPLE",
        "sampleInterval": "10000000000"
      }
    ],
    "encoding": "JSON_IETF"
  }
}
```

Once the JSON RPC has been sent, the device window shows the requested telemetry.

```
update {
  timestamp: 1649702177210584000
  update {
    path {
      origin: "rfc7951"
      elem {
        name: "Cisco-IOS-XE-interfaces-oper:interfaces"
      }
      elem {
        name: "interface"
      }
    }
    val {
      json_ietf_val: "[{\\"name\\":\\"AppGigabitEthernet1/0/1\\",\\"interface-type\\":\\"iana-"}"
    }
  }
}
```

Processing returns...

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": "2022-01-13T15:09:21.833000+00:00",
    "if-index": 49,
    "phys-address": "70:1f:53:9b:0f:a9",
    "speed": "1000000000",
    "statistics": {
      "discontinuity-time": "2022-01-13T15:06:19+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": 4238849808,
      "out-unicast-pkts": "39319036",
      "out-broadcast-pkts": "15106020",
      "out-multicast-pkts": "23408303",
      "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": "4238849808"
    }
  }
}
```

gRPC dial-out telemetry receiver plugin

YANG Suite also has a gRPC receiver plugin that can be used to receive the telemetry data and display it in the GUI. The data can also be sent to a log file on the disk as well as to the Elasticsearch database, where it can be stored and later used for visualizations and alerting.

Listen at IP address: Listen at port: (Optional) TLS receiver

```
timestamp: 2022 Apr 11 18:49:48:525000
subscription: 57598
node: jcohoe-c9300
subscribe_path: /Cisco-IOS-XE-process-cpu-oper:cpu-usage/cpu-utilization
child_path: /
name: five-seconds
value: 2
```

Telemetry Receivers ✕

Output file

Elasticsearch Output URI

IP Address	Port	TLS
<input type="text" value="0.0.0.0"/>	<input type="text" value="57598"/>	<input type="checkbox"/> false

Tooling: Cisco Crosswork

Cisco Crosswork Suite

Cisco Crosswork is designed with modern low-touch and no-touch operations in mind. It is multivendor and multidomain, centered on both programmatic infrastructure control and access to operational and state data, and spans both cloud/SaaS and on-premises tooling. The tools in the portfolio encompass the full-service lifecycle and deliver a closed operational loop that includes planning and design, implementation, and ongoing monitoring and assurance.

While elements of the Crosswork portfolio can be licensed individually, Crosswork is also available packaged in two suite options. These suites group tools that share a common purpose and also provide the advantage of lower licensing costs than if you purchased the tools individually.

Learn more about the entire Crosswork Suite options here:

<https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/crosswork-network-automation/crosswork-essentials-adv-suites-ds.html>

Cisco Crosswork Data Gateway

Cisco Crosswork Data Gateway has been developed for real-time data collection from multivendor network devices. This application simplifies the collection challenges of all this network traffic.

Cisco Crosswork Data Gateway is an on-premises application deployed close to network devices, enabling multiple data collection methods—model-driven telemetry, SNMP, CLI, etc. The collected data is delivered securely and consumed by on-premises and cloud analytics applications. Cisco Crosswork Data Gateway enables a critical and important tenet of data collection: the collection process should be an efficient and centralized step.

Cisco Crosswork Data Gateway assumes the job of connecting to devices, collecting the data, and publishing it. Pushing a first-stage processing function closer to the source data, Crosswork Data Gateway can reduce the amount of data sent to the application and the stress on the devices, abstracting the network complexity and reducing application vendor dependencies. If the use case dictates, Cisco Crosswork Data Gateway can send the raw data for direct consumption by the registered application.

Instances of Cisco Crosswork Data Gateway can be distributed to support large-scale networks. Crosswork applications scale better by offloading data collection and processing to distributed Data Gateway instances closer to the devices.

Learn more about Cisco Crosswork Data Gateway here:

<https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/crosswork-network-automation/datasheet-c78-743287.html>

Tooling: Cisco Telemetry Broker

Cisco Telemetry Broker has roots in the Stealthwatch UDP Director (UDPD), which simply replicated UDP traffic to multiple destinations. The Cisco Telemetry Broker builds upon the successes of the UDPD while also creating a new Telemetry Broker market. Cisco Telemetry Broker optimizes telemetry pipelines for the hybrid cloud. It vastly 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.

The benefits of Cisco Telemetry Broker include brokering, filtering, and transforming data. This provides the ability to route and replicate telemetry data from a source location to multiple destination consumers, to filter data that is being replicated to consumers for fine-grained control over what consumers are able to see and analyze, and to transform data protocols from the exporter to the consumer's protocol of choice.

Learn more about Cisco Telemetry Broker here:

- Cisco Telemetry Broker <https://cs.co/telemetrybroker>
- Read more in the blog: <https://blogs.cisco.com/security/taking-full-control-of-your-telemetry-with-the-intelligent-telemetry-plane>

Tooling: The TIG stack

Telegraf, InfluxDB, and Grafana (TIG) make up the TIG stack, which is an open-source set of software that can be used to receive, store, and visualize model-driven telemetry data. Each software component can scale horizontally to enable distributed collection and storage as well as centralized analysis, processing, and alerting. Many integrations and plugins are available with this set of software for flexible deployments and unique use cases and requirements.

Telegraf, InfluxDB, and Grafana stack

Telegraf is tooling that handles data collection and processing. It is responsible for dialing into or receiving data from the model-driven telemetry interfaces on Cisco IOS XE. Telegraf has a rich ecosystem of plugins that allow it to receive and process data from a variety of places, as well as to send the data out to an equally rich ecosystem of plugins.

InfluxDB is used as the data storage or database layer. Telegraf pushes the telemetry data it receives into the API of Influx, where the data eventually resides. Once the data is within InfluxDB, it can be queried, visualized, and made sense of by network analysts and operators.

Grafana is used for data visualizations—it makes API calls into Influx where the data is in order to visualize some of the time series data into charts, graphs, panels, and other visually pleasing methods of representing complex or raw data points.

Together Telegraf, InfluxDB, and Grafana make a simple yet powerful stack of software components that is used to receive, process, store, and visualize telemetry data. TIG is just one example; there are many other tools, both commercial and open source, that can be leveraged for the same purposes. Often Telegraf is integrated into existing telemetry and data lake collection systems to receive network telemetry and pass it into existing systems like Splunk or PowerBI.

TIG stack with Docker

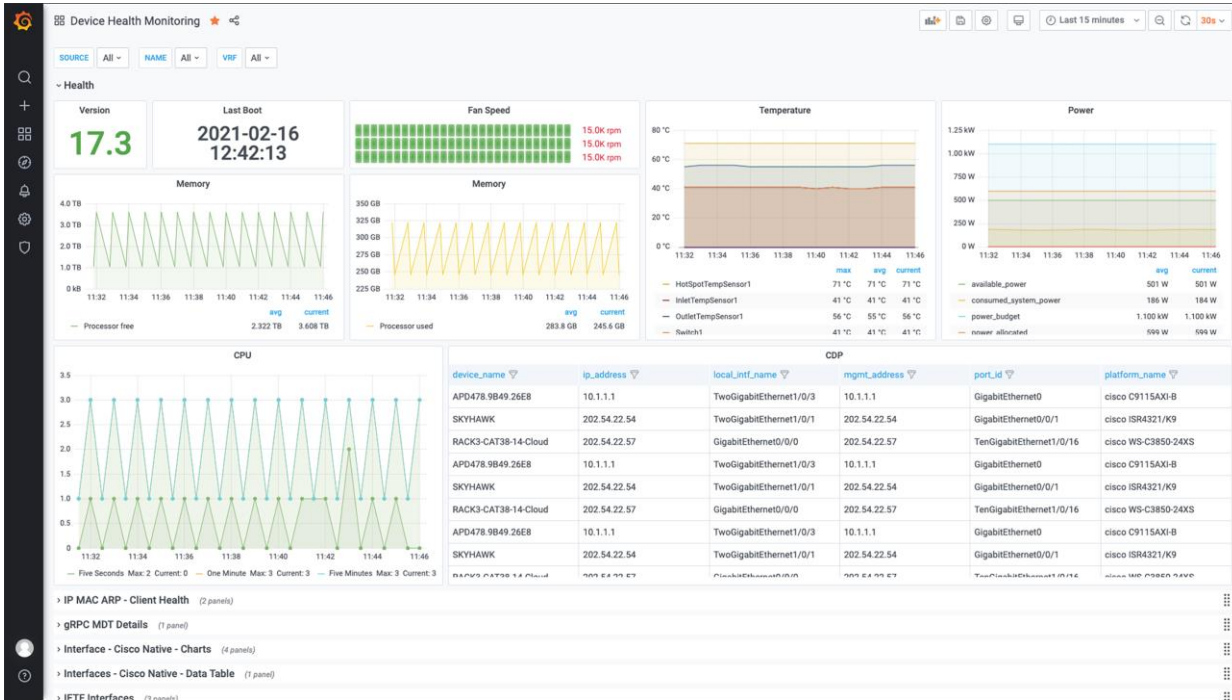
In production scenarios, the components of the TIG stack are deployed in a custom configuration tailored to the use cases and requirements of the business and network. An example configuration has been provided within the Docker container. This container is easy to deploy to get started with the TIG stack components using the various model-driven telemetry interfaces.

- TIG Stack with Docker example: <https://github.com/jeremycohoe/cisco-ios-xe-mdt>

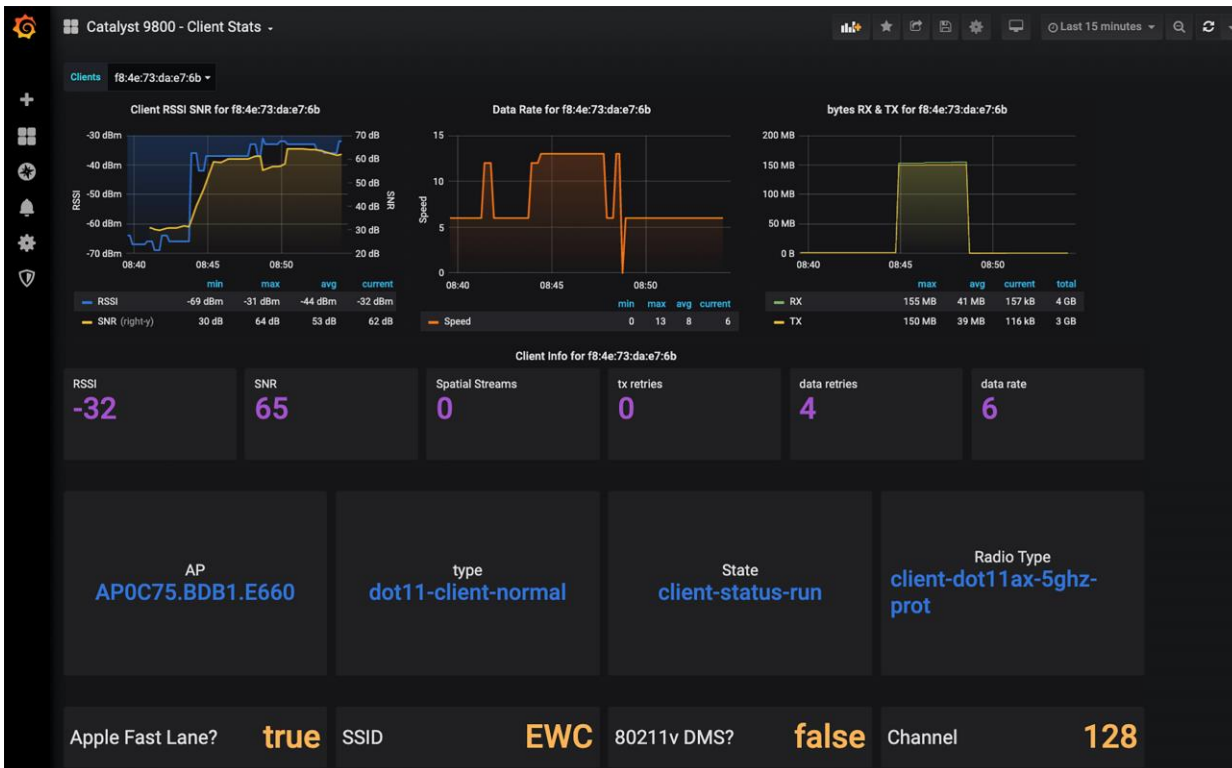
Grafana dashboard examples

The example Grafana dashboards are available to show end-to-end use cases for visualizing the telemetry data over time.

Device health monitoring: <https://grafana.com/grafana/dashboards/13462>



Wireless client stats: <https://grafana.com/grafana/dashboards/12468>

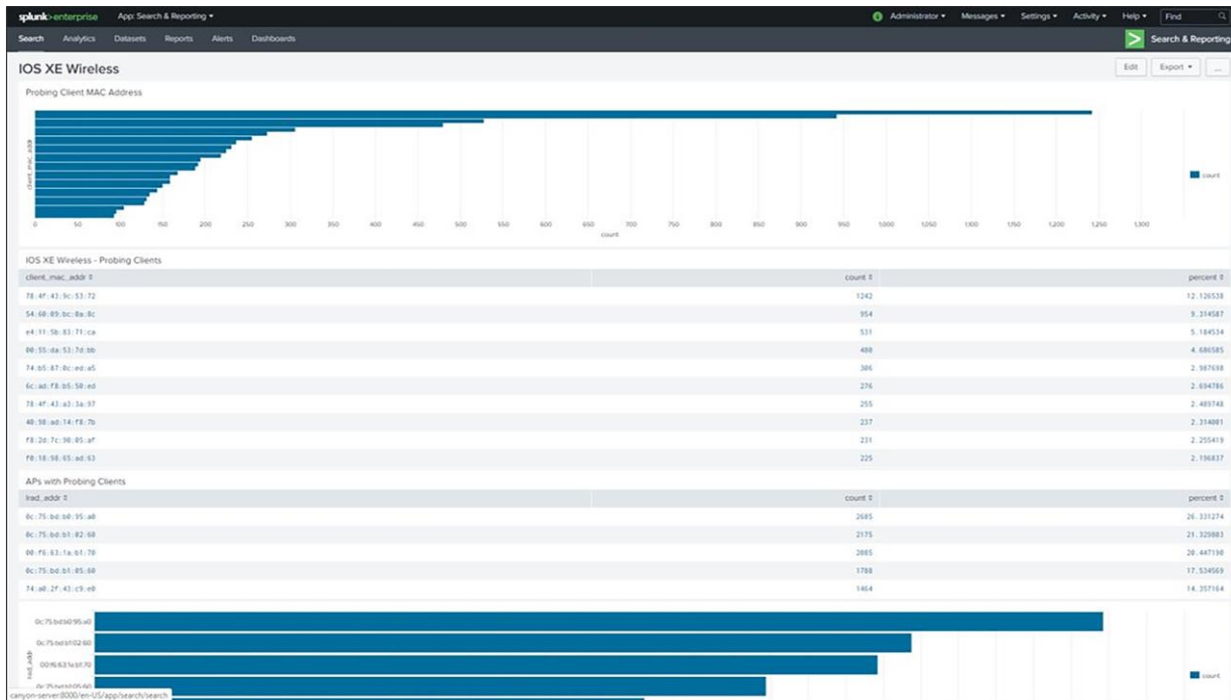


Splunk integration

While YANG Suite is often used for validation of telemetry, when it becomes time to move into production, several options are available to process, store, and act on the telemetry data that is received. Splunk is software that is commonly used for this purpose and is seen in many environments for many different use cases, from applications to networks.

Examples for integrating model-driven telemetry with Splunk are available from the GitHub and Splunk links:

- <https://github.com/jeremycohoe/cisco-ios-xe-mdt/blob/master/telegraf-splunk.conf>
- https://www.splunk.com/en_us/blog/it/the-daily-telegraf-getting-started-with-telegraf-and-splunk.html



Day N: Device optimization

There are many features as part of the device optimization capabilities, including Guest Shell and gNOI integrations to ensure that day N device optimization can be completed as needed, which ensures that the various customizations and optimizations can be performed.

Guest Shell is a virtualized CentOS Linux-based environment that is designed to run custom scripts and applications, including Python, for increased automation, control, and management capabilities. It can be used to install, update, and operate third-party Linux applications and custom scripts. Guest Shell is bundled with the system image and can be enabled and configured as required. It has integrations to the Cisco IOS XE CLI and NETCONF/YANG API and can also be used programmatically and automatically when integrated with the Cisco IOS Embedded Event Manager (EEM) feature.

gNOI is a workflow API that is part of the gNMI programmatic interface. It uses protocol buffers, or protobuf for short, and there are several “.proto” definitions for various workflows that have been defined within the openconfig/gnoi repository on GitHub. Several of these.proto workflows have been implemented, including the certificate management API service (cert.proto), the operating system software management API service (OS.proto), and the reset to factory API service (reset.proto), and several more are being considered. Tooling for each of these gNOI implementations is available from the google/gnxi repository on GitHub and include gnoi_cert for cert.proto, gnoi_os for os.proto, and gnoi_reset for reset.proto, among others.

Learn more:

- <https://github.com/openconfig/gnoi>
- <https://github.com/google/gnxi>

The CLI to YANG feature helps convert the running config command into YANG format for the programmatic interfaces. In addition to reading the Cisco-IOS-XE-native.yang configuration from the API, this command can be used to easily retrieve the running config in either XML for NETCONF or for RESTCONF formatted in JSON. This is an easy way, when on the command line, to see the YANG configuration that is in the show running config command. This XML or JSON formatted code can more easily be used within third-party automation and orchestration tooling systems and controllers.

gNOI

gNOI is a workflow API that is part of the gNMI programmatic interface. Once gNMI is enabled, the gNOI workflows also become available and are accessed through the same API service. Protobufs are an encoding technique used to structure the data. Several protobufs are implemented and many more defined within the OpenConfig/gnoi Github repository. Refer to the section “Cisco Native RPC Actions YANG” for additional information about operations and workflows.

The .proto workflows that have been implemented include:

- The certificate management API service (cert.proto)
- The operating system software management API service (OS.proto)
- The reset to factory API service (reset.proto)

Tooling for each of these gNOI implementations includes:

- gnoi_cert for cert.proto
- gnoi_os for os.proto
- gnoi_reset for reset.proto

gNOI certificate management service

The gNOI cert.proto certificate management service can be used to install cryptographic certificates into the trustpoint for use within services and applications on Cisco IOS XE. The trustpoint is an abstract container that is used to store certificates used for secure communications between a client and a server. The gnoi_cert client tooling is used to install or revoke certificates on Cisco IOS XE from third-party controllers, scripts, and orchestration systems.

gNOI certificate bootstrapping is a feature within cert.proto for when a target device does not have any preexisting certificates. The certificate bootstrapping feature allows the installation of certificates for the purpose of establishing subsequent secure gNMI and thus secure gNOI connections.

Essentially, when gNMI is enabled with the bootstrapping “secure-init” command, the first certificate that gNOI cert.proto receives is installed into the trustpoint and is also applied to the gNMI service. As part of this operation, the gNMI service is restarted with the newly installed certificate to accept subsequent secure connections with signed certificates.

The following gNOI cert.proto operations allow complete certificate management services:

- Install
- Rotate
- Revoke
- Get
- GenerateCSR

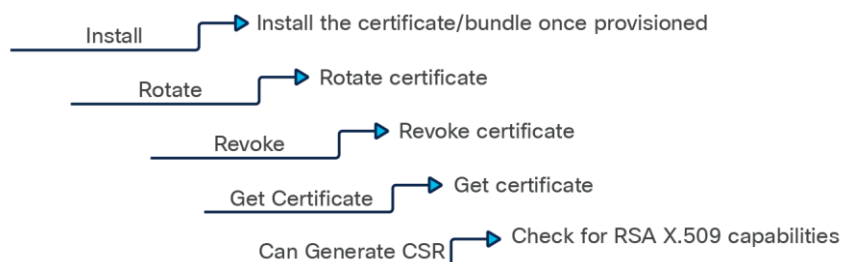


Figure 27.
gNOI cert.proto operations

The certificate management service has been defined in the OpenConfig consortium’s gNOI repository in GitHub at <https://github.com/openconfig/gnoi/blob/master/cert/cert.proto> and has tooling available from Google’s gNxl repository on GitHub at https://github.com/google/gnxi/tree/master/gnoi_cert

gNOI operating system installation service

The gNOI OS.proto operating system service is used to manage the currently running operating system version and has operations that include install, activate, and verify. Together these operations are used to programmatically upgrade or downgrade the Cisco IOS XE between releases, for example, to upgrade from 17.5 to 17.6, from 17.6.1 to 17.6.2, or even downgrades such as moving from 17.8 to 16.8 if needed.

Bundle mode or install mode can be configured when booting software images, and OS.proto works regardless of which boot mode was configured. However, when the Cisco IOS XE device is booted in bundle mode, it will be converted to install mode as part of the programmatic gNOI OS install and activate operations. Install mode has several advantages over bundle mode for high availability, device boot, and reload times.

The operating system installation service has been defined in the OpenConfig consortium's gNOI repository in GitHub at <https://github.com/openconfig/gnoi/blob/master/os/os.proto> and has tooling available from Google's gNxl repository on GitHub at https://github.com/google/gnxi/tree/master/gnoi_os.

Details about the gNOI operating system installation service can be retrieved using the verify operation, but are also available from a traditional show command.

```
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
```

gNOI factory reset service

The gNOI reset.proto factory reset service is used to perform factory reset operations to bring devices back to a new state as if just received from the factory. The primary operation is the “start” RPC, which starts the “factory-reset all” or “factory-reset switch all all” operations against the target device. There is one additional option as part of this RPC, which is zero fill.

The factory reset service is supported in install mode but not in bundle mode. If bundle mode is in use, the start API call will be rejected and will not complete. The gNOI OS.proto operating system installation service can be used in this case when bundle mode is used to upgrade and reboot into install mode.

The factory reset service has been defined in the OpenConfig consortium's gNOI repository at https://github.com/openconfig/gnoi/blob/master/factory_reset/factory_reset.proto and has tooling available from Google gNxl repository on GitHub at https://github.com/google/gnxi/tree/master/gnoi_reset.

Details about the gNOI factory reset service can be retrieved with the show gNxl command.

```
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
```

The gNOI microservices enable much easier management of certificates, operating systems, and the factory reset process.

CLI to YANG

The CLI to YANG feature helps convert the running config command into YANG format, either for NETCONF with XML or RESTCONF with JSON. CLI to YANG requires the netconf-yang data model interfaces to be enabled starting with Release 17.7. Commands with corresponding native YANG and modeled in show run are returned.

Format for NETCONF with XML encoding

Convert commands to NETCONF XML as well as for RESTCONF with XML encoding.

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

```
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>
```

Format for RESTCONF with JSON encoding

Convert commands to RESTCONF JSON:

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

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

Additionally, we can filter for specific portions of the running config using either Netconf-xml or restconf-json:

```
c9300-pod29#show run vrf | format restconf-json
{
  "data": {
    "Cisco-IOS-XE-native:native": {
      "vrf": {
        "definition": [
          {
            "name": "Mgmt-vrf",
            "address-family": {
              "ipv4": {
                "ip": "10.10.10.10",
                "mask": "255.255.255.0"
              },
              "ipv6": {
                "ip": "2001:db8::1",
                "mask": "ffff:ffff:ffff:ffff::"
              }
            }
          }
        ]
      },
      "interface": {
        "GigabitEthernet": [
          {
            "name": "0/0",
            "shutdown": [null],
            "vrf": {
              "forwarding": "Mgmt-vrf"
            },
            "Cisco-IOS-XE-ethernet:negotiation": {
              "auto": true
            }
          }
        ]
      }
    }
  }
}
c9300-pod29#

c9300-pod29#show run aaa | format netconf-xml
<config xmlns="http://tail-f.com/ns/config/1.0">
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <username>
      <name>admin</name>
      <privilege>15</privilege>
      <secret>
        <encryption>9</encryption>
        <secret>$9$kV9UHURDpKJHsk$j05q/C4ZtgnWFqYJ3AcFSAYsCoRhWaMNZDCui1f0t2</secret>
      </secret>
    </username>
  </native>
  <aaa>
    <new-model xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-aaa"/>
      <session-id xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-aaa">common</session-id>
    </aaa>
  </config>
c9300-pod29#
```

Guest Shell

Guest Shell is a virtualized Linux-based environment, designed to run custom Linux applications, including Python, for automated control and management of Cisco devices. Using Guest Shell, you can also install, update, and operate third-party Linux applications. Guest Shell is bundled with the system image and can be installed using the “guestshell enable” Cisco IOS command. This container shell provides a secure environment, decoupled from the host device, in which users can install scripts or software packages and run them. The existing network hardware is used to deliver the scalability, high availability, and flexibility required, with no requirements for dedicated or separate compute. The Guest Shell environment is intended for tools, Linux utilities, and manageability rather than networking.

Guest Shell and application hosting

Guest Shell is a built-in CentOS container that has APIs into Cisco IOS XE via Python and NETCONF, which uses the Cisco IOx infrastructure. The same IOx infrastructure is used for Docker-based application hosting, which enables an even wider range of Linux features that can be used. Application hosting, like Guest Shell, can be completely managed with the YANG APIs. The configuration of the feature and the operational state of the containers can be managed with the YANG model. Controller solutions with Cisco DNA Center also enable management and integration of the ThousandEyes performance monitoring solution. Refer to the [Application Hosting White Paper](#) and DevNet resources at <https://developer.cisco.com/docs/app-hosting/> for more details on application hosting.

Guest Shell feature enablement

Before enabling Guest Shell, IOx must be configured. If IOx is not configured, a message to configure IOx is displayed. Removing IOx removes access to Guest Shell. To enable and operate Guest Shell, the management interface needs to be configured on the device. To enable IOx, enter the following commands:

```
configure terminal
iox
exit
```

Enabling Guest Shell on the management interface:

```
configure terminal
app-hosting appid <name>
app-vnic management guest-interface <interface number>
end
show app-hosting list
```

Once the prerequisite configuration is set up, enable and enter the Guest Shell container:

```
guestshell enable
```

The output will look like the following:

```
C9300# conf t
Enter configuration commands, one per line. End with CNTL/Z.
Device(config)# app-hosting appid guestshell
Device(config-app-hosting)# app-vnic management guest-interface 1
Device(config-app-hosting-mgmt-gateway)# end
C9300# show app-hosting list
App id                               State
-----
guestshell                             DEPLOYED

C9300# guestshell enable
Interface will be selected if configured in app-hosting
Please wait for completion
```

```
guestshell activated successfully
Current state is: ACTIVATED
guestshell started successfully
Current state is: RUNNING
Guestshell enabled successfully
```

```
C9300#show app-hosting list
```

App id	State
-----	-----
guestshell	RUNNING

Guest Shell resources

Resources used by the Guest Shell container can be checked with the following CLI command. The hardware resource allocations for CPU, memory, and disk are displayed. Persistent disk space can also be increased as needed.

```
C9300# show app-hosting utilization appid guestshell
```

```
c9300-pod21#show app-hosting utilization appid guestshell
Application: guestshell
CPU Utilization:
  CPU Allocation: 800 units
  CPU Used:      0.00 %
  CPU Cores:    0-7
Memory Utilization:
  Memory Allocation: 256 MB
  Memory Used:      66320 KB
Disk Utilization:
  Disk Allocation: 1 MB
  Disk Used:       0.00 MB
c9300-pod21#
```


Verifying Guest Shell

To confirm that the IOx service has been enabled, enter the `show iox-service` command and ensure that the IOx Cisco application hosting framework (CAF), IOx service (IOxman), and Libvirt are in the running state.

```
C9300# show iox-service

IOx Infrastructure Summary:
-----
IOx service (CAF)           : Running
IOx service (HA)           : Running
IOx service (IOxman)       : Running
IOx service (Sec storage)  : Running
Libvirt 5.5.0              : Running
Docker v19.03.13-ce       : Running
Sync Status                 : Disabled
```

```
C9300# show app-hosting list
```

```
App id                               State
-----
guestshell                            RUNNING
```

Accessing and using Guest Shell

Linux commands can be run directly from the IOS CLI. The `guestshell run Bash` command opens the Guest Shell Bash prompt. To log into Guest Shell, run the following command:

```
auto@pod21-xelab:~$ telnet c9300
Trying 10.1.1.5...
Connected to c9300-pod21.
Escape character is '^]'.

User Access Verification

Username: admin
Password:

c9300-pod21#gu
c9300-pod21#guestshell
[guestshell@guestshell ~]$ id
uid=1000(guestshell) gid=1000(guestshell)
[guestshell@guestshell ~]$
```

```
C9300# guestshell
[guestshell@guestshell ~]$ pwd
/home/guestshell
[guestshell@guestshell ~]$ whoami
guestshell
[guestshell@guestshell ~]$ uname -a
Linux guestshell 5.4.69 #1 SMP Fri Mar 19 21:47:56 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

Guest Shell with Python API

Python scripts can be run in Guest Shell. The “guestshell run python3” commands launch the Python interpreter.

```
C9300# guestshell run python3
```

Once the interactive shell is entered, the “cli” Python module can be used to execute commands as needed.

Cisco IOS CLI commands:

```
>>> from cli import clip
>>> clip("show ip int brief | exclude unassigned")
```

```
c9300-pod21#guestshell run python3
Python 3.6.8 (default, Dec 22 2020, 19:04:08)
[GCC 8.4.1 20200928 (Red Hat 8.4.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import clip
>>> clip("show ip int brief | exclude unassigned")
Interface                IP-Address      OK? Method Status        Protocol
GigabitEthernet1/0/24    10.1.1.5        YES other  up            up
Loopback0                 192.168.12.1    YES other  up            up
>>>
```

Noninteractive Python

Guest Shell can execute Python scripts in a noninteractive environment. Files can be placed on the device bootflash at day 0 during ZTP provisioning or at any other time during the device lifecycle. Scripts can be copied from a network TFTP service or created manually when required using common tooling and editors like “vi.” The script can be executed using the “guestshell run python3 show.py” command, which can be used in other integrations, including EEM, the Embedded Event Manager.

```
[guestshell@guestshell ~]$ cat show.py
from cli import clip
clip("show ip int brief | e una")
exit()

[guestshell@guestshell ~]$ python3 show.py
Interface                IP-Address      OK? Method Status        Protocol
GigabitEthernet1/0/24    10.1.1.5        YES other  up            up
Loopback0                 192.168.12.1    YES other  up            up
```

```
#!/usr/bin/python

from cli import clip
clip("show ip int brief | exclude unassigned")
exit()
```

The command to execute the Python script is:

```
guestshell run python3 /home/guestshell/show.py
```

```
c9300-pod21#guestshell
[guestshell@guestshell ~]$ pwd
/home/guestshell
[guestshell@guestshell ~]$ ls
show.py
[guestshell@guestshell ~]$ exit
exit

c9300-pod21#guestshell run python3 show.py
Interface      IP-Address      OK? Method Status      Protocol
GigabitEthernet1/0/24  10.1.1.5        YES other  up          up
Loopback0      192.168.12.1   YES other  up          up

c9300-pod21#
```

Guest Shell with the Cisco IOS CLI

The `dohost` command is built into Guest Shell and will send the command directly to the device. The command is limited to `exec` privilege mode and is not for the `config` mode.

```
[guestshell@guestshell ~]$ dohost 'show ip int brief | e una'
Interface      IP-Address      OK? Method Status      Protocol
GigabitEthernet1/0/24  10.1.1.5        YES other  up          up
Loopback0      192.168.12.1   YES other  up          up

[guestshell@guestshell ~]$
```

Guest Shell with NETCONF API

Guest Shell Python runs in an LXC container. This container is managed by IOx, which is a container, similar in function to Docker, managed specifically for Cisco IOS XE.

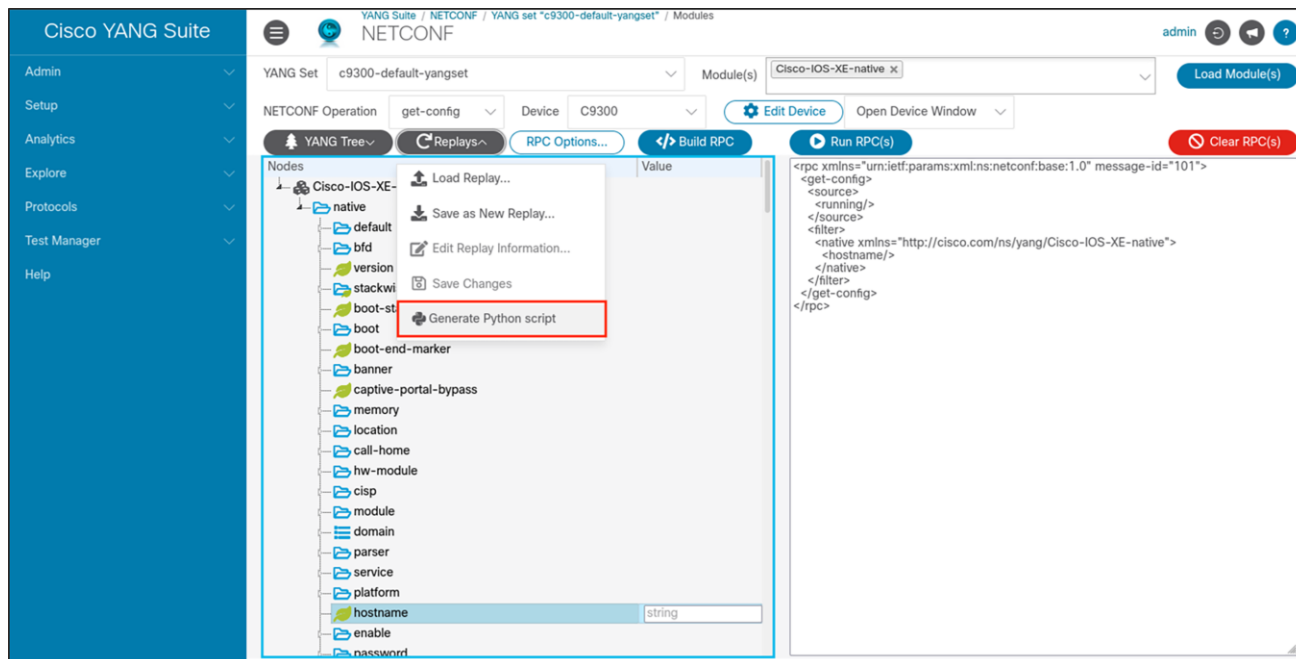
Before using Guest Shell, enable IOx and then enable Guest Shell. Additionally, ensure that the following is configured to open the bridge between Guest Shell and Cisco IOS XE:

```
netconf-yang ssh local-vrf guestshell enable
```

Passwordless authentication using keys can easily be set up using the following Bash commands or the Python API "netconf_enable_guestshell"

```
[guestshell@guestshell iosp_client -f netconf_enable guestshell 830  
[guestshell@guestshell iosp_client -f netconf_enable_passwordless guestshell guestshell
```

The following Python script was generated using YANG Suite and returns the hostname of the given device.



get_hostname.py

```
#!/usr/bin/env python  
  
from ncclient import manager  
import sys  
import xml.dom.minidom  
  
HOST = '127.0.0.1'  
# use the NETCONF port for your device  
PORT = 830  
# use the user credentials for your device  
USER = 'guestshell'  
PASS = 'it will use the key specified and does not use this one here ok'  
  
FILTER = '''  
    <filter xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```

        <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
            <hostname></hostname>
        </native>
    </filter>
'''

def main():
    """
    Main method that prints netconf capabilities of remote device.
    """
    # Create a NETCONF session to the router with ncclient
    with manager.connect(host=HOST, port=PORT, username=USER,
                        password=PASS, hostkey_verify=False,
                        device_params={'name': 'default'},
                        key_filename="/home/guestshell/.ssh/id_rsa_netconf",
                        allow_agent=False, look_for_keys=True) as m:

        # Retrieve the configuration
        results = m.get_config('running', FILTER)
        # Print the output in a readable format
        print(xml.dom.minidom.parseString(results.xml).toprettyxml())

if __name__ == '__main__':
    sys.exit(main())

print("\n\n *** Finished NETCONF example... *** \n\n")

```

Run this script from Guest Shell using the following command. Any block of XML that is generated from YANG Suite can easily be used in place of the simple “get hostname” example above.

```
c9300-pod21# guestshell run python3 netconf.py
```

```

c9300-pod21#guestshell run python3 netconf.py
<?xml version="1.0" ?>
<rpc-reply message-id="urn:uuid:141753ab-20c6-4e63-b664-68aef8b8755f" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
      <hostname>c9300-pod21</hostname>
    </native>
  </data>
</rpc-reply>

c9300-pod21#

```

The above example shows that Guest Shell has made a connection to the localhost 127.0.0.1 on port 830, which is connected to Cisco IOS XE's NETCONF interface for management and operation data use cases.

Guest Shell integration with EEM

Embedded Event Manager (EEM) is a distributed and customized approach to event detection and recovery offered directly in a Cisco IOS device. EEM offers the ability to monitor events and take informational, corrective, or any desired EEM action when the monitored events occur or when a threshold is reached. An EEM policy is an entity that defines an event and the actions to be taken when that event occurs.

EEM can be used to execute Python scripts within the Guest Shell environment. In this example, whenever a syslog message is generated indicating that an interface has changed state from up to down or into a disabled state, the guestshell_script.py Python script will run. This script runs some CLI commands and saves the output to a log file.

The example EEM applet below is used to log messages from the CLI and YANG interfaces to syslog. This helps in debugging and logging and can be shipped off-box if necessary for security and auditing use cases.

```

enable
configure terminal
event manager applet catchall
event cli pattern ".*" sync no skip no
action 1 syslog msg "$_cli_msg"
end

```

Disabling and destroying Guest Shell

The “guestshell disable” command shuts down and disables Guest Shell.

The “guestshell destroy” command removes the rootfs from the flash filesystem.

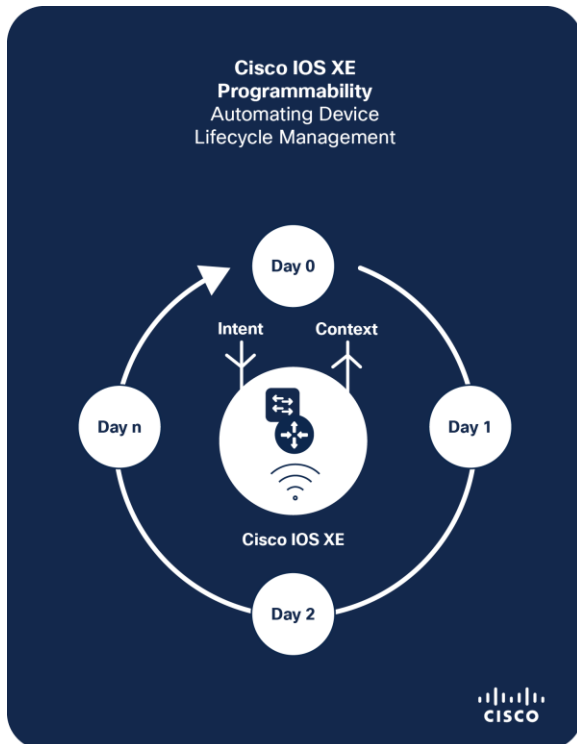
Guest Shell conclusion

As described in this section, Guest Shell is a very powerful integration option when used in conjunction with other features and technologies like EEM, Python3, the CLI, NETCONF APIs, and more.

Conclusion

The Cisco IOS XE network OS delivers an innovative level of programmability and automation, decreasing the complexity of the business and network. This white paper has described the need for programmable interfaces and the differences between them for the full device lifecycle, including device onboarding, configuring, monitoring and optimization. The programmatic interfaces NETCONF, RESTCONF, and gRPC can be enabled and configured for communicating with Cisco devices. Configured and dynamic telemetry subscriptions can be established using open-source tools. Example payloads can be used to create, verify, and remove a feature programmatically using the YANG Suite, Ansible, and Terraform tooling. We have learned the various ways to work with Cisco IOS XE YANG models programmatically. We can automate any Cisco IOS XE device using any interface. As the needs of network engineers are ever evolving, we at Cisco will also continue to provide the cutting-edge technologies needed to shape the future.

Additional resources



- Cisco IOS XE Programmability Book: <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 Configuration Guide: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/176/b_176_programmability_cg.html?dtd=osscdc000283
- YANG Suite: <https://developer.cisco.com/yangsuite/>
- NCC: <https://github.com/CiscoDevNet/ncc>
- Embedded Event Manager: <https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/eem/configuration/xe-17/eem-xe-17-book.html?dtd=osscdc000283>
- NETCONG YANG Configuration and Validation: <https://www.cisco.com/c/en/us/support/docs/storage-networking/management/200933-YANG-NETCONF-Configuration-Validation.html>

Developer community and feedback

- Cisco DevNet: <https://developer.cisco.com/>
- Cisco Communities: <https://community.cisco.com>
- Cisco DevNet for Cisco IOS XE: <https://developer.cisco.com/site/ios-xe/>
- Cisco IOS XE model-based management docs: <https://developer.cisco.com/docs/ios-xe/#!/model-based-management-introduction>
- MDT Learning Lab: https://developer.cisco.com/learning/modules/iosxe_telemetry
- Automation Exchange: <https://developer.cisco.com/network-automation/>

Blogs

- Zero-touch provisioning: <https://blogs.cisco.com/developer/device-provisioning-with-ios-xe-zero-touch-provisioning>
- Model-driven telemetry: <https://blogs.cisco.com/developer/getting-started-with-model-driven-telemetry>
- Cisco IOS XE automation: <https://blogs.cisco.com/ciscoit/b-en-04162014-look-ma-no-hands?dtd=osscdc000283>
- SNMP to model-driven telemetry: <https://blogs.cisco.com/developer/its-time-to-move-away-from-snm-and-cli-and-use-model-driven-telemetry>
- Terraform Cisco IOS XE provider: <https://blogs.cisco.com/developer/terraformiosxe01>

Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters
Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters
Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at <https://www.cisco.com/go/offices>.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <https://www.cisco.com/go/trademarks>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)