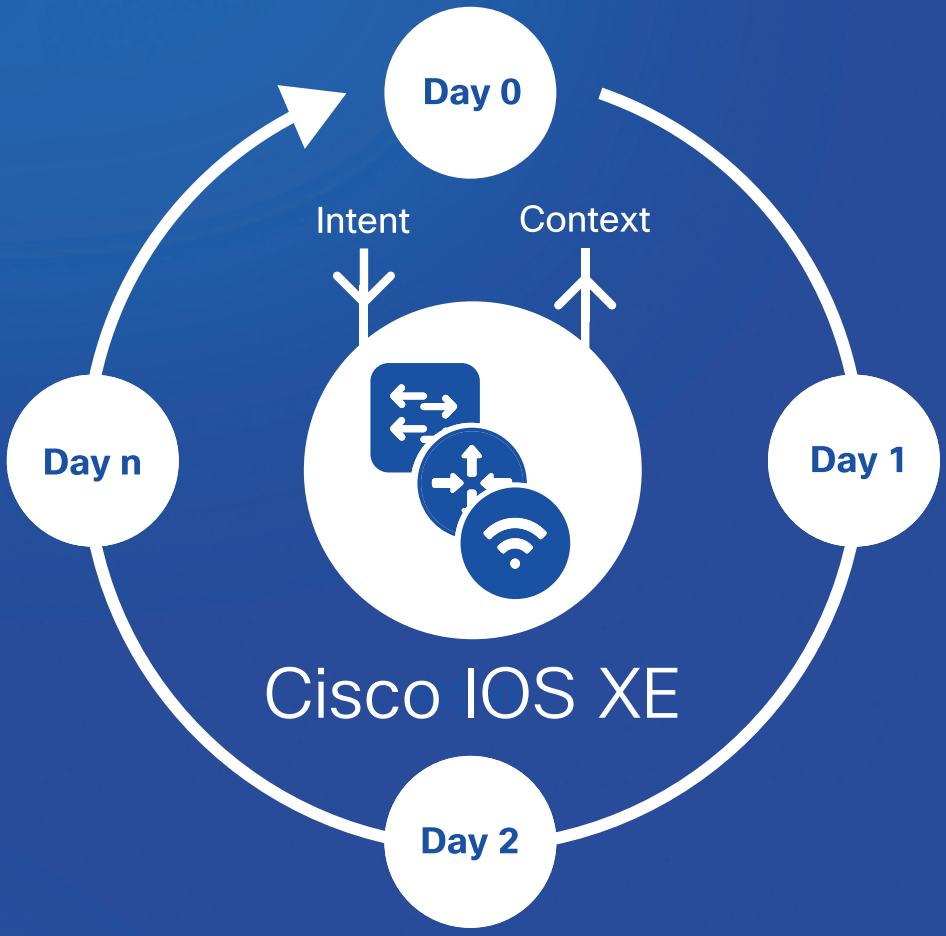
Cisco IOS XE Programmability

Automating Device Lifecycle Management



cisco

Adam Radford
Fabrizio Maccioni
Gabriel Zapodeanu
Itai Koren
Jeff McLaughlin
Jeremy Cohoe
Jonathan Yang
Krishna Kotha
Nabil Michraf
Robert Grasby



Cisco IOS XE Programmability

Automating Device Lifecycle Management

Table of Contents

Authors 7

Acknowledgments 8

About this Book 9

Introduction 12

Why Programmability Matters 13

Lifecycle of Network Device Operations 14

Use Cases 16

Operational Approaches 18

Next Steps 24

General Concepts 25

Cisco IOS XE 26

What is Programmability? 28

Application Programming Interfaces (APIs) 30

Programming Languages 31

Structured Data 32

Data Encoding Formats 34

Day 0 Device Onboarding 36
Introduction 37
Zero-Touch Provisioning (ZTP) Scenarios 41
Basic ZTP Workflow 42
Advanced ZTP Workflows 44
Considerations 47
Next Steps 48

YANG 49
Overview 50
YANG Concepts 54
YANG Native vs Open Data Models 58
YANG Data Model Highlights 61
YANG Tools 64

Network Device APIs 68
Overview 69
NETCONF 70
RESTCONF 75
Comparison of NETCONF and RESTCONF 78
Next Steps 79

Telemetry 80
Overview 81
Operational Data 82
Flow Data 85
Use Cases 86
Subscription Tools 88
Data Collectors 89

Python 92
Overview 93
Python WebUI Sandbox 96
On-Box Python 98
Advanced On-Box Python 100
Common Issues 103

Guest Shell 104
Introduction 105
Security 106
Configuration and Updates 107
Resource Allocation 108
Use Cases 110
Next Steps 111

Application Hosting 112
Introduction 113
Cisco Application-Hosting Framework 115
Containers and Virtual Machines 121
Use Case 123
Next Steps 124

Controllers 125
Introduction 126
Common Controllers 128
Why Use a Controller? 130

DevOps and NetDevOps 131
Introduction 132
Continuous Integration and Delivery 134
DevOps Tools 136
Next Steps 139

Appendices 140
Additional Resources 141
Acronyms 145

Authors

This book is the result of a collaborative effort across Cisco's Technical Marketing, Product Management, Engineering, and Sales teams. The individual authors are:

- Adam Radford – Distinguished Systems Engineer
- Fabrizio Maccioni – Technical Marketing Engineer
- Gabriel Zapodeanu – Technology Solutions Architect
- Itai Koren – Product Manager
- Jeffery McLaughlin – Principal Technical Marketing Engineer
- Jeremy Cohoe – Technical Marketing Engineer
- Jonathan Yang – Technical Leader
- Krishna Kotha – Technical Marketing Engineer
- Nabil Michraf – Solutions Architect
- Robert Grasby – Product Manager

Acknowledgments

A special thanks to Cisco's Enterprise Networking Business Product Management, Engineering, Sales and Services teams who supported the realization of this book. Thanks to Carl Solder and Muninder Sambi for their support; Christina Munoz and Cynthia Resendez for exceptional logistics; and Sehjung Hah for his assistance in coordinating this effort.

We would also like to thank:

- Andrew Hobby
- Andrew Brown

We are also genuinely appreciative of our Book Sprints team:

- Faith Bosworth (facilitator)
- Henrik van Leeuwen and Lennart Wolfert (illustrators)
- Juan Gutierrez (development)
- Julien Taquet (book producer)
- Sue Tearne and Raewyn Whyte (copyeditors)

A Book Sprint (*https://www.booksprints.net/*) is a strongly facilitated process for collaborative authorship of books.

About this Book

Network engineers are tasked with deploying, operating and monitoring the network as efficiently, securely and reliably as possible.

As the complexity, scale and other demands (such as security) of the network increase, there is greater pressure on network engineers to reduce time to implement and minimize the risk of changes.

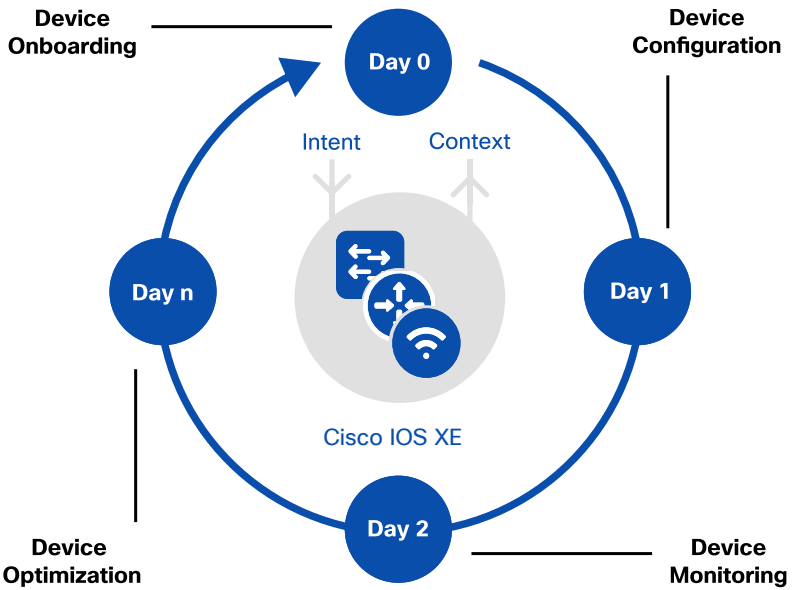
Automating Device Lifecycle Management with Cisco IOS® XE Programmability

The Cisco IOS® XE network Operating System (OS) is the single OS for enterprise switching, routing, wired and wireless access.

It provides open, standards-based programmable interfaces to automate network operations and brings deep visibility into users, applications and device behaviors.

Automating device lifecycle management through IOS XE programmability, as shown in the following diagram , assists network engineers to reduce business and network complexity.

DIAGRAM *Lifecycle Management*



Automation Benefits

Automating device lifecycle management provides the following benefits:

- determinism and greater network insight
- increased business agility and productivity
- lower operational costs

Operational Approaches

This book addresses the various operational approaches to integrate a network element programmatically with controllers, DevOps tools and direct connection to devices. The Cisco IOS XE network OS has been designed to enable all three integration options.

The features covered in this book are available in IOS XE 16.6 and above.

Intended Audience for this Book

This book provides a summary of the general concepts of programmability supported by Cisco IOS XE. Many books on programmability assume the reader has a back ground in software development and understands fundamental concepts about how software is built. This book does not make these assumptions. Instead, it is primarily written for network engineers who are new to programmability and wish to under stand the basic concepts before moving on to advanced topics. For engineers with some programmability experience, this book will summarize the specific capabilities of Cisco IOS XE.

Introduction

Why Programmability Matters

The demands associated with operating enterprise networks today are increasing as businesses undergo digital transformation and increasingly adopt the Internet of Things (IoT), mobile, and cloud-based services.

The network has become integral to business operations, and as a result, Information Technology (IT) teams are being required to:

- deploy new infrastructure and business services faster
- increase network reliability and security
- reduce costs associated with operating the network

In response to these challenges, Cisco has been reinventing the way campus and Wide Area Networks (WAN) are built, deployed and operated. At the heart of this change is a move from network operations based on manual changes, to an automated approach. In short, it is a new paradigm where hardware and software are used to build the network.

The new era of enterprise networks requires an open and flexible network Operating System (OS) that provides open, standards-based programmable interfaces to automate network operations and brings deep visibility into user, application, and device behaviors.

The Cisco IOS XE network OS addresses these needs as the single OS for enterprise switching, routing, wired and wireless access. It delivers a transformational level of automation and programmability, reducing business and network complexity.

This book outlines the programmability capabilities in Cisco IOS XE and the benefits for network engineers.

Lifecycle of Network Device Operations

Network engineers are tasked with deploying, operating and monitoring the network as efficiently, securely and reliably as possible.

The first challenge is getting a device onto the network. This is commonly referred to as "Day 0" device onboarding. The key requirement is to get the device connected with as little effort as possible. Depending on the operational mode and security requirements, either a small subset of configuration or a "full" initial configuration will be deployed.

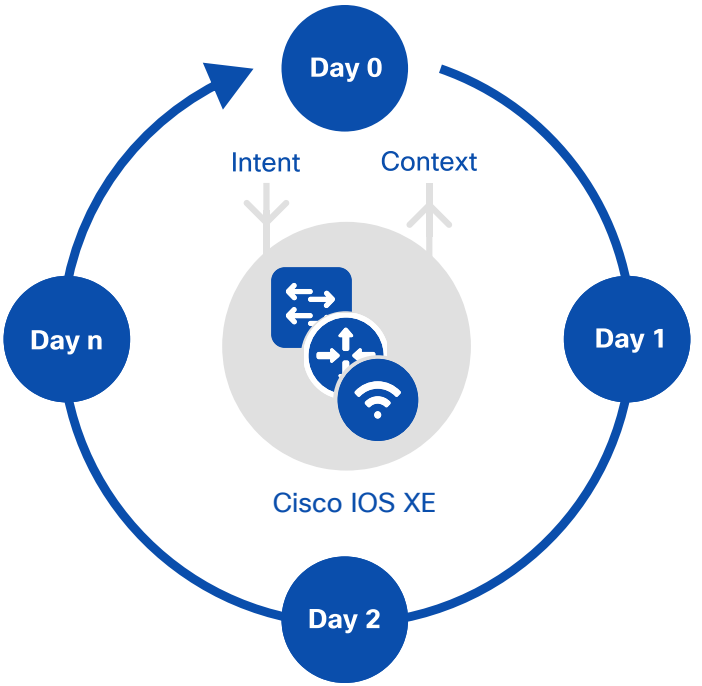
Once the device is provisioned, the configuration of the device needs to be maintained and upgraded. "Day 1" device configuration management is responsible for the ongoing configuration. Changes to the configuration need to be reliable, efficient, and auditable.

The next challenge is monitoring the network and measuring performance. "Day 2" device monitoring provides a view of how the network is operating. Based on this view, some changes may be required.

Lastly, optimizations to the device are made, such as extensions to the device capabilities, enhancements to operational robustness, or patches and software up grades. Referred to as "Day n", this implies an iterative approach, looping back through Days 0, 1, and 2.

This lifecycle is captured in the diagram below.

DIAGRAM *Device Lifecycle*



As the complexity, scale, and other demands of the network increase, there is greater pressure on network engineers to reduce time to implement and minimize the risk of changes.

Automation offers two distinct benefits:

- **Determinism:** automated changes provide predictable results while manual processes do not.
- **Speed of change:** automated changes are faster as human input is not required.

These two factors are important for improving operational efficiency.

Use Cases

Some examples of the benefits of leveraging Cisco IOS XE programmability are described below.

Day0–DeploymentAutomation

Network device onboarding is a manual and time-consuming operation, requiring highly skilled engineering personnel. Onboarding is tedious and repetitive in nature. Network engineers automate the entire provisioning workflow by using the Cisco IOS XE Zero-Touch Provisioning (ZTP) feature. Automated device onboarding with programmable workflows lowers the cost and time required to provision a network device, eliminating errors and allowing the use of lower level engineering personnel and associated resources.

Day1–CloudServiceProvisioningAutomation

Network devices located in dynamic environments, such as the cloud, need to avoid being the bottleneck in service provisioning. Rapid and repeatable service delivery is crucial in this environment. The feature-rich Cisco IOS XE device Application Programming Interfaces (API) enable efficiently automated configuration changes.

Dayn–Optimization

Events such as link flaps, power supply failure, and configuration drift are difficult to act on in a consistent manner. The Cisco IOS XE Guest Shell feature assists IT organizations to develop an entirely new suite of applications that help with existing operational challenges. Python scripts running on Guest Shell can trigger alerts to IT Network Operation Centers (NOC) when new critical events are detected, automatically create service tickets, and mitigate the issues by dynamically applying configurations.

For example, oil and gas producers have many remote locations where Internet or WAN access bandwidth is very limited and expensive. Data collected from drilling operations needs to be centralized in a corporate data center. Cisco IOS XE Application Hosting provides a solution to host the compression applications at the edge of the network so the data can be compressed to consume less bandwidth.

Operational Approaches

There are three operational approaches to programmatically integrate a network element:

- via a controller such as Cisco's Digital Network Architecture Center (DNA Center)
- via a configuration management tool (i.e. DevOps)
- directly to the device

Each comes with various benefits and trade-offs. However, the Cisco IOS XE network OS has been designed to enable all three integration options.

Controller Integration

Through controller integration, programmatic control of the underlying network elements is abstracted via an intermediary to simplify automation efforts.

Controllers are purpose-built, exposing only a subset of the underlying network's element features and functionality, as the underlying capabilities are abstracted through the controller. Controllers usually expose Representational State Transfer (REST) APIs for northbound integration. Their southbound interfaces may not be based on open protocols, potentially limiting integration with Cisco IOS XE's standards-based network configuration interfaces.

Some controllers such as Cisco DNA Center are designed to provide closed-loop feedback, allowing the controller to dynamically adjust network configurations based on changing network context.

DIAGRAM *Controller Integration to Cisco IOS XE Devices*

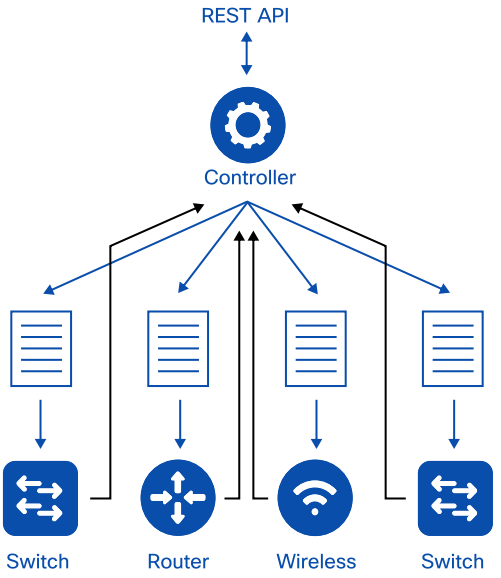
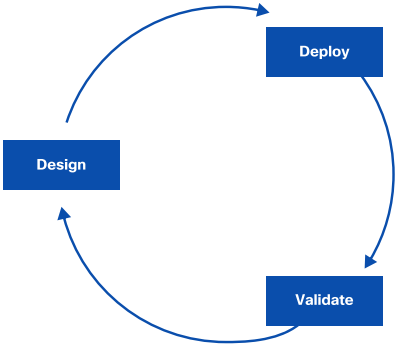


DIAGRAM *Controller Configuration Change Workflow*



Key considerations when integrating with controllers include support for:

- atomic network configurations: confirm that the intended configuration has been applied to all network elements without error
- auditing and managing configuration drifts: identify when a network element's configuration has changed from its intended state
- limiting and blocking direct device configuration changes: prevent configuration drift

For more information, see the chapter on Controllers.

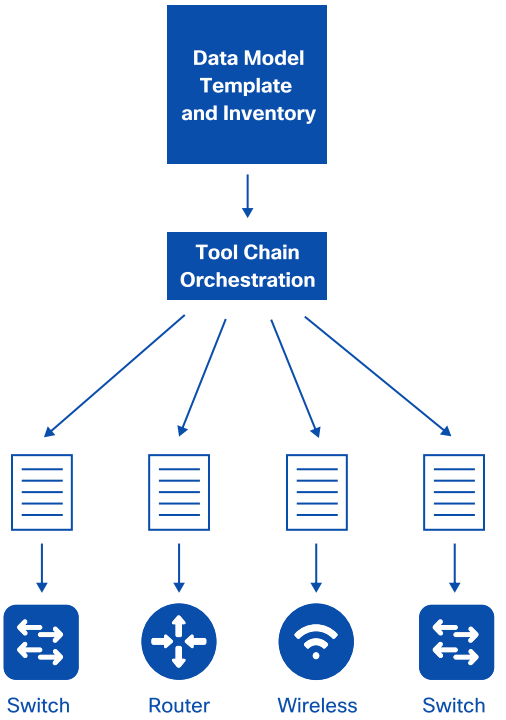
Configuration Management Tool Integration

Configuration Management tools enable DevOps workflows and access to the full feature set of the device. In a DevOps workflow configuration, changes are "modeled" and run through comprehensive validation in a simulation environment prior to deployment.

Configuration management tools are used not only to manage network devices but also to manage compute and application resources. Their input is in the form of a simplified data model to provide human readability. Their southbound interfaces may not be based on open protocols, and this could limit integration with Cisco IOS XE's standards-based network configuration interfaces.

Configuration management tools do not provide closed-loop feedback. Instead, configuration changes are extensively tested and validated in a simulation environment before being pushed into production. Validation testing and configuration pushes are orchestrated through Continuous Integration (CI) toolchains.

DIAGRAM *Configuration Management Tool Integration to Cisco IOS XE Devices*



Key considerations when integrating via DevOps are similar to integrating via a controller, specifically support for:

- atomic configurations
- auditing and managing configuration drifts
- limiting and blocking direct device configuration changes

For more information see the chapter on DevOps and NetDevOps.

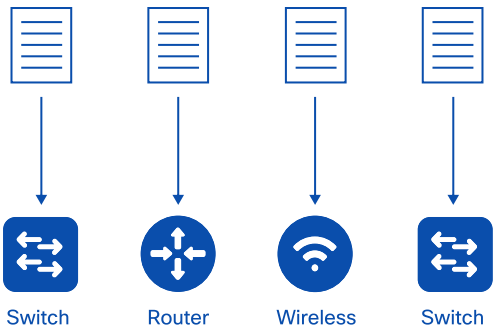
DIAGRAM *DevOps Configuration Change Workflow*



Direct Integration

Direct integration, as the name implies, involves a direct programmatic control of each network element. While manageable with a small number of devices, this approach is more challenging in networks with more devices.

DIAGRAM *Direct Integration to Cisco IOS XE Devices*



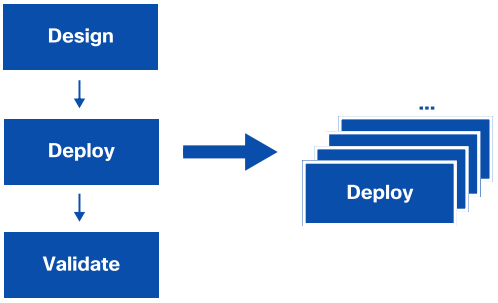
Cisco IOS XE devices support RFC 6241 NETCONF and RFC 8040 RESTCONF network configuration protocols, providing the option of XML-based or JSON-based integration.

With direct integration, configuration changes are made on a single set of devices, then replicated across the entire network.

Direct integration is helpful for monitoring network devices and ensuring the changes made did not create undesired behaviors.

Direct integration is helpful for monitoring network devices and ensuring the changes made did not create undesired behaviors.

DIAGRAM *Direct Integration Configuration Change Workflow*



Key considerations with direct integration include:

- scale: managing changes to all applicable devices
- change validation: ensuring changes made do not have undesired effects

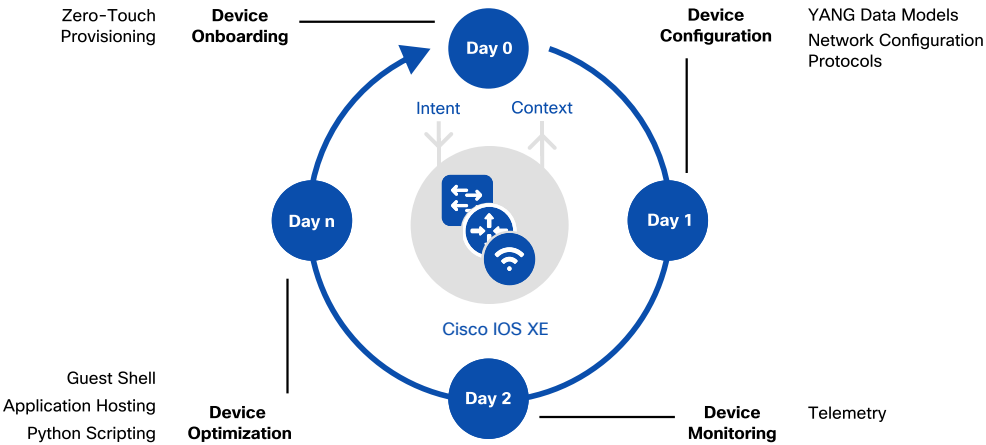
Next Steps

Automated interaction with the infrastructure is required, no matter which operational approach is chosen, either controller, DevOps or direct.

Cisco IOS XE is architected from the ground up to automate all phases of the operational lifecycle, from Day 0 device onboarding, through configuration and monitoring, to optimization.

The next chapters map to each phase of the device lifecycle. The last two chapters of the book cover operational approaches based on Controllers and DevOps tools.

DIAGRAM *Device Lifecycle Phases*



General Concepts

Cisco IOS XE

Cisco IOS XE is an open and flexible network operating system optimized for the new era of intent-based enterprise networks.

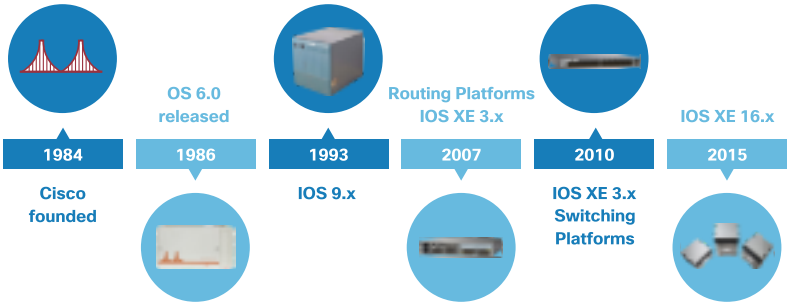
Recommended Release

The features covered in this book are available in Cisco IOS XE 16.6 and above.

Cisco IOS XE History

Cisco Systems was founded in 1984, long before the Internet as we know it existed. Since then, Cisco's hardware platforms and network operating systems have evolved. Cisco's original Internetwork Operating System (IOS) formed the backbone of computer networks worldwide for many years. Cisco IOS XE represents an evolution of Cisco's operating system, providing a solid foundation for modern programmable interfaces and data models.

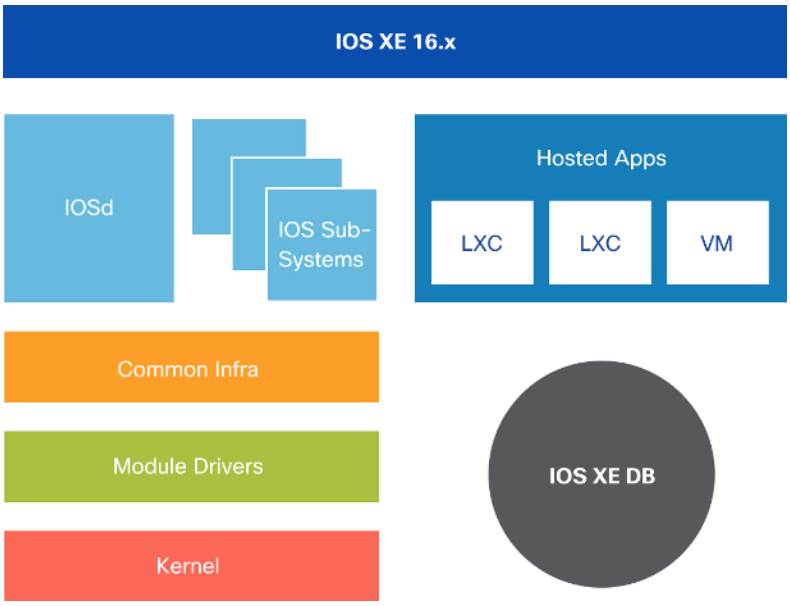
DIAGRAM *Cisco IOS XE Evolution*



Cisco IOS XE Architecture

The Cisco IOS XE architecture allows feature components to run as subsystems over common infrastructure, module drivers, and kernel. Flexibility is given to feature components such as Border Gateway Protocol (BGP). The architecture supports Linux Containers (LXC) and Virtual Machines (VM) to be securely hosted on the network device. IOS XE provides a system database for both configuration and operational data state accessible through programmatic interfaces.

DIAGRAM *Cisco IOS XE Architecture*



What is Programmability?

Programmability Overview

Routers, switches, firewalls, and wireless controllers have historically been managed directly by humans. Management interfaces for interacting with these devices are human-oriented and are not designed or intended for machine consumption. Configurations are applied to these devices via the Command-line Interface (CLI) or via a Graphical User Interface (GUI). It is possible to automate these interfaces but because they are designed for humans, machine automation is highly inefficient.

Programmability is the capability to configure and manage networking devices using software protocols. Unlike human-oriented interfaces such as CLI and GUIs, programmable interfaces are designed specifically to be consumed by machines. For the pur-poses of this book, programmability also refers to capabilities that are supported on Cisco IOS XE devices to automate interactions with network devices and technologies including Guest Shell and Application Hosting. These machine interfaces allow rapid device configuration, easing the burden of network administration to enable new ways for network operators to obtain operational data, which makes analyzing network behavior easier.

CLI to Programmability Transition

Programmability will not be an immediate replacement for CLI-based and GUI-based management of network devices. Programmability will automate many of the more tedious and error-prone tasks typically done via CLI. Using programmability to manage a network requires a period of learning and transition on the part of network operators. Managing a network with programmable interfaces requires a significantly different approach than traditional CLI.

The best way to begin this transition is to identify a few specific tasks or processes that can be automated. For example, for a network engineer applying the same configuration to a large number of devices, using model-based configuration is easier then box-by-box CLI. An operational task that is performed repeatedly through out

the day by Network Operation Center (NOC) engineers is more easily per formed by a script. The Cisco IOS XE native data models are aligned with CLI, making them easily understood by experienced network engineers. Tools making the CLI to program mability transition easier are referenced throughout this book.

Application Programming Interfaces (APIs)

An Application Programming Interface (API) is a set of routines, protocols, and tools that integrate software components. A Network API provides automation and programmability which connects software to network devices. A network element is considered programmable if it exposes an API that can be used by software components to program the element.

These APIs may access configuration (writable) data or operational (read-only) data. The interfaces can be standards-based or proprietary.

An API supported by a programmable device needs to be documented correctly, so the consumer of the API can use it appropriately.

Programming Languages

Network engineers are looking for human-friendly scripting languages with adequate related libraries. Python, Ruby and Go are the most commonly used. Python is the most popular of the scripting languages for network automation.

Python

Python is a user-friendly interpreted (executed line-by-line) high-level scripting language with a large and comprehensive set of libraries. Python interpreters are available for all major operating systems.

Ruby

Ruby is a dynamic, open source scripting language with a focus on simplicity and productivity.

Go

Go is an open source programming language created by Google, and is behind a number of cloud-centric projects.

Structured Data

Structured data is the key to building programmable interfaces which follow consistent, predictable, and well-defined patterns. The following is an example of unstructured data:

```
John  415555121225  Main Street
```

This unstructured data contains different elements that are not clearly delineated, and it is not clear where each element begins and ends. While a human may be able to recognize and parse this data, a machine will not.

Below is an example of the same data in a structured format:

```
Name: John
Phone: 415-555-1212
Address: 25 Main Street
```

Each line consists of a key-value pair delineated by a colon. In the example above, "Phone:" is the key and "415-555-1212" is the value.

Are CLI Outputs Structured Data?

At first glance, it would appear that CLI outputs are structured, as they seem to be presented with identifying keys. The following example of the CLI command **sh int e1/10** output:

```
switch1#sh int e1/10
Ethernet1/10 is up
  Hardware: 1000/10000 Ethernet, address: 0005.73d0.9331 (bia 0005.73d0.9331)
  Description: To UCS-11
  MTU 1500 bytes, BW 1000000 Kbit, DLY 10 usec,
      reliability 255/255, txload 1/255, rxload 1/255
  Switchport monitor is off
  EtherType is 0x8100
  Last link flapped 8week(s) 2day(s)
  Last clearing of "show interface" counters 1d02h
  30 seconds input rate 944 bits/sec, 118 bytes/sec, 0 packets/sec
  30 seconds output rate 3110376 bits/sec, 388797 bytes/sec, 5221 packets/sec
```

While the data is nicely formatted with spacing and indentation, there is no obvious delimiter for key-value pairs. For example, the Media Access Control (MAC) address has a key called "address:" with the associated value. However, apart from "Description:" there is no other obvious key-value pair.

Structured data ensures that key-value pairs and their hierarchy are easily identifiable. YANG data models are an example of a specification for structured data. These data models can be used to generate XML-formatted and JSON-formatted payloads which apply to configuration and operational data (as will be described in the following sections of this book).

Data Encoding Formats

Configuration and operational data are represented as key-value pairs. The key identifies what the data is, while the value is the actual data. For example, when configuring a Layer 3 interface with an IP address of 10.10.10.10, "IP address" is the key, while "10.10.10.10" is the value.

It is important that both the client and the server understand the encoding format of the data. XML, JSON and YAML are commonly used encoding formats for representing the configuration and operational data on a device.

eXtensible Markup Language (XML)

XML is markup language for describing data in a way that is both human and machine-readable. Much like HTML, XML uses tags to identify data. Tags are enclosed by angle brackets (" <>") and the tags (key) surround the data (value)

JavaScript Object Notation (JSON)

JSON is an encoding format originally developed for JavaScript. JSON was intended to provide a human and machine-readable encoding format that can be used as a replacement for XML. JSON uses a key-value representation with a ":" as a delimiter instead of the opening and closing tags used by XML.

YAML Ain't Markup Language (YAML)

YAML is a human-readable data-serialization language. It is commonly used for configuration files but could be used in many applications where data is being stored or transmitted.

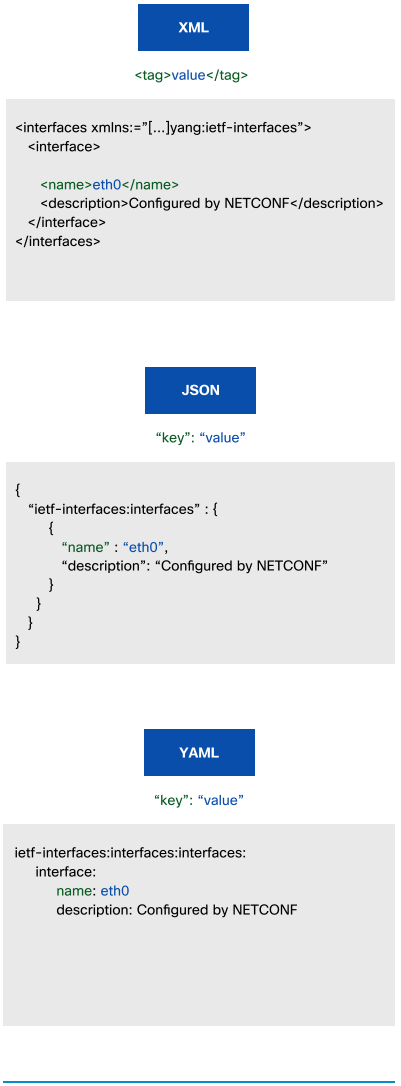
Note that in the XML example, the values, such as "eth0", are enclosed by an opening and closing tag. The closing tag is denoted by a forward slash. XML also can show hierarchy by nesting tags.

In the JSON example, there are no tags. The key-value pairs are enclosed by quotation marks (" "), and are separated by a colon (:). JSON also can show hierarchy by using parentheses.

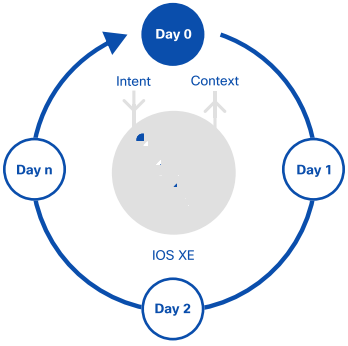
In the YAML example, again there are no tags. The key-value pairs are separated by a colon (:). YAML represents the hierarchy by using spaces and carriage returns, unlike XML and JSON.

It is important to note that XML, JSON, and YAML serve the same purpose. They are simply different ways of doing the same thing. The encoding format is defined by the protocol.

DIAGRAM *XML, JSON and YAML Comparison*



Day 0 Device Onboarding



Introduction

Automated device onboarding ensures that network devices are put into service using pre-defined, managed configurations which are applied programmatically. In a traditional setting, onboarding is a manual process carried out by skilled individuals. Network engineers would go on site, connect, and configure the device. The process is quite manual and error-prone.

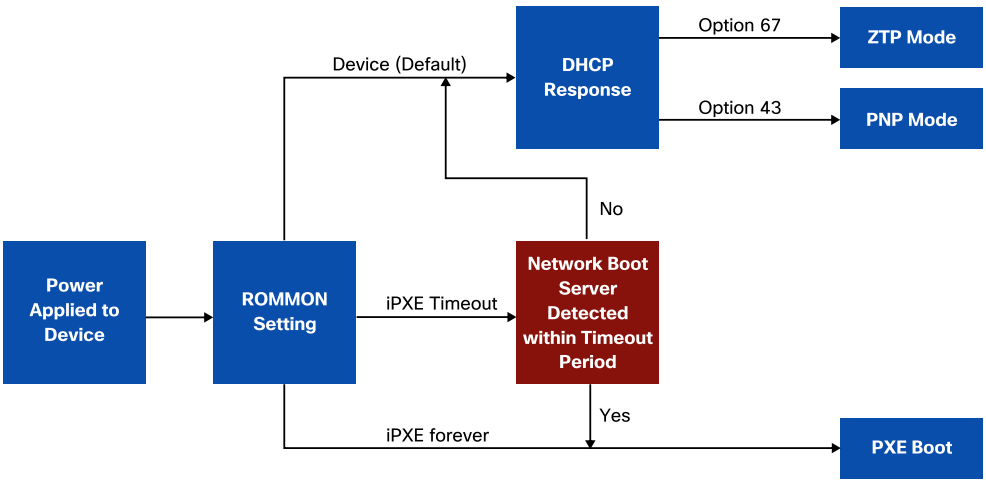
With automated device onboarding, individuals with no networking knowledge or expertise can install the physical device. Initial configurations are deployed program automatically, based on pre-defined business intent – reducing costs, avoiding human errors and improving business agility. This provides a simple, secure, and integrated option to ease new branch or campus rollouts.

Cisco IOS XE supports the following capabilities to automate device onboarding: Zero-Touch Provisioning (ZTP), Cisco Network Plug and Play (PnP), and Pre-Boot Execution Envi- ronment (PXE).

Boot Modes

The IOS XE boot mode is based on a Read-Only Memory Monitor (ROMmon) setting.

DIAGRAM *Cisco IOS XE Boot Modes*



Device Mode

By default, ROMmon is set to "device" which boots using the local image.

When the power is applied to a Cisco IOS XE network device, the device boots from the hosted operating system. If there is no configuration file present, the device uses Dy- namic Host Control Protocol (DHCP) to determine whether the device should enter ZTP or Network PNP automated provisioning mode, based on options in the DHCP offer. The DHCP options available for these modes are 67 and 43.

Zero-Touch Provisioning (ZTP) Mode

ZTP helps to automate the process of installing or upgrading software images and installing configuration files on Cisco devices that are deployed in a network for the first time. When a device boots up and does not find the startup configuration, the device enters the Zero-Touch Provisioning mode. The device locates a DHCP server, boot-straps itself with its interface IP address, gateway, and Domain Name System (DNS) server IP address, and enables Guest Shell. The device then obtains the IP address or URL of an HTTP or TFTP server through DHCP Option 67 and downloads the Python script to configure the device.

Guest Shell provides the environment for the Python script to run. Guest Shell executes the downloaded Python script which applies an initial configuration to the device.

Zero-Touch Provisioning provides an open and flexible device onboarding workflow. The network operator is responsible for all aspects of this workflow, including building, testing and validating the Python script as well as hosting it on an appropriate server.

Network Plug and Play (PnP) Mode

Upon receiving DHCP Option 43, the device initiates a connection to the PnP server such as DNA Center. This triggers a pre-defined workflow that can install a certificate, upgrade the Cisco IOS XE software version, and apply the initial configuration to the device.

The PnP server can pre-provision the network device: Device rules are created in advance to map the serial number of the device to the appropriate workflow. The workflow contains elements such as IOS XE version, network configuration, and exec commands to run on the device.

If there is no matching rule for the network device on PnP, the device will appear as an "unclaimed". The operator can then assign the network device to a workflow and continue the provisioning process.

Preboot eXecution Environment (PXE) Mode

When the device is in PXE mode, it boots from an image hosted on an external server. The PXE server is automatically detected by the Cisco IOS XE device, using the standard PXE interface. The IOS XE image is then downloaded using HTTP, FTP, or TFTP protocols.

There are two PXE boot options:

- **PXE Timeout:** If the PXE process fails and the timeout expires, the default device boot is activated.
- **PXE Forever:** The Cisco IOS XE device will keep sending DHCP requests forever until it receives a DHCP response, without falling back to the default device boot.

Zero-Touch Provisioning (ZTP) Scenarios

The most common scenarios for ZTP are:

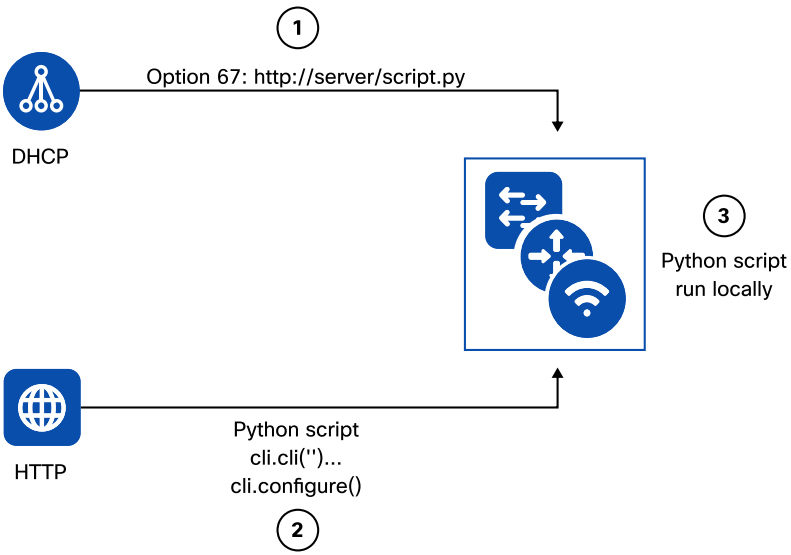
- **Out-of-the-Box:** A new device is deployed on the network and needs a configuration file
- **Return Material Authorization (RMA):** A device has failed and is being replaced by a similar device, the configuration file of the failed device needs to be applied to the replacement
- **Adds/moves/changes:** Before a device is redeployed, it needs to be reset to factory defaults, which moves it to the "Out-of-the-Box" mode

As mentioned earlier, the ZTP process only begins if the device has no configuration. The **write erase** command is used to remove the configuration on a device so the ZTP process can begin again.

Basic ZTP Workflow

In the simplest scenario, the device gets an IP address, contacts the HTTP/TFTP server, and downloads a Python script. The Python script will configure the device with a set of credentials or configure an authentication server, then the device can be managed by other tools.

DIAGRAM *Basic ZTP Workflow with HTTP Server*



In the diagram above, the first step is to get an IP address via DHCP. The DHCP offer will also include option 67, pointing to a Python script on an HTTP server. The network device will download the Python script and execute these commands locally on Guest Shell.

A basic Python configuration script appears below. This script sets the hostname, the username and password, and the enable secret which allows local login on the Virtual

Terminal Line (VTY) ports. A remote user can then connect to the device to manage it. Since the CLI Python library has a direct connection to the device, no authentication is required. Other methods besides CLI are also available.

```
from cli import configure, execute
USER="cisco"
PASSWORD="cisco"
ENABLE="cisco"

def base_config():
    configure(['hostname ztp-server'])
    configure(['username {} privilege 15 password {}'.format(USER,PASSWORD)])
    configure(['enable secret {}'.format(ENABLE)])
    configure(['line vty 0 4', 'login local'])
    execute('write memory')

base_config()
```

The device configuration is saved to Non-Volatile Random-Access Memory (NVRAM) using the **write memory** command to ensure persistence. If this step is omitted and the device reboots, it will go back to the ZTP process. There are some scenarios where this behavior is desirable, as will be seen in the advanced section.

Common Issues

These are common issues encountered when working with ZTP:

- Some IOS XE devices only support ZTP on the management port
- Access to the TFTP or HTTP ports on the server should be permitted from the network device

Advanced ZTP Workflows

Dynamic Configuration Use Case

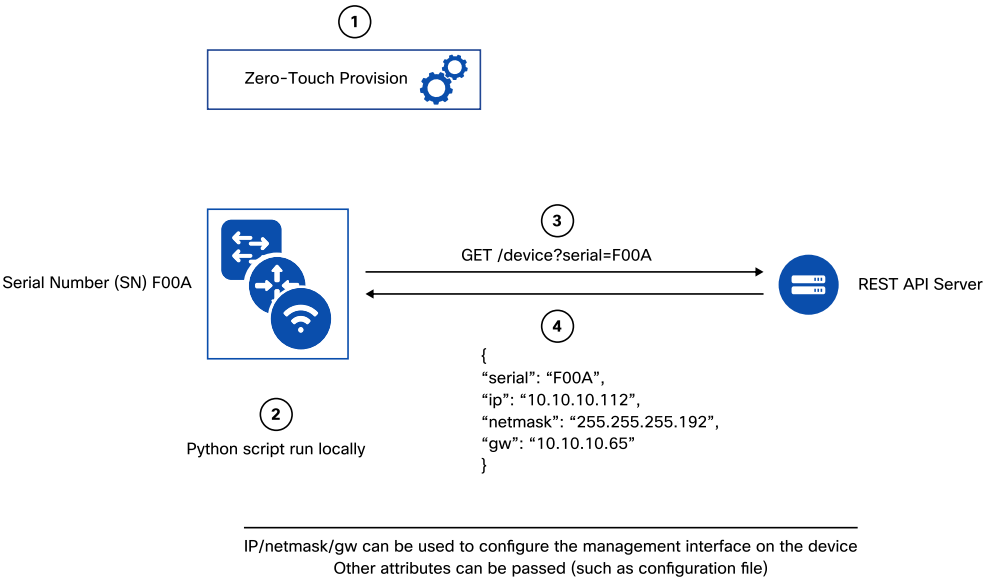
The simple scenario referenced earlier used a simple static configuration file. Often there are parts of the configuration that are dynamic and include variables. The Python script can use a REST API call to obtain the dynamic variables. A common scenario is setting the management interface, netmask, and gateway.

The Python script calls a REST API, passing the serial number as a parameter. The REST server uses the device serial number to look up a set of pre-defined parameters such as IP address, netmask, and gateway. The Python script uses these variables in the configuration applied to the device.

Coding the REST server is outside the scope of this book. Many open source frameworks are available to build simple APIs.

This workflow could be extended further by passing a link to a Jinja template to populate local variables from the device, such as the number of interfaces on a switch. Jinja is one of many popular templating languages that can be used for this purpose. These templates could be stored in a git repository with version control.

DIAGRAM *Advanced ZTP Workflow with REST APIs*



Stacking Use Case

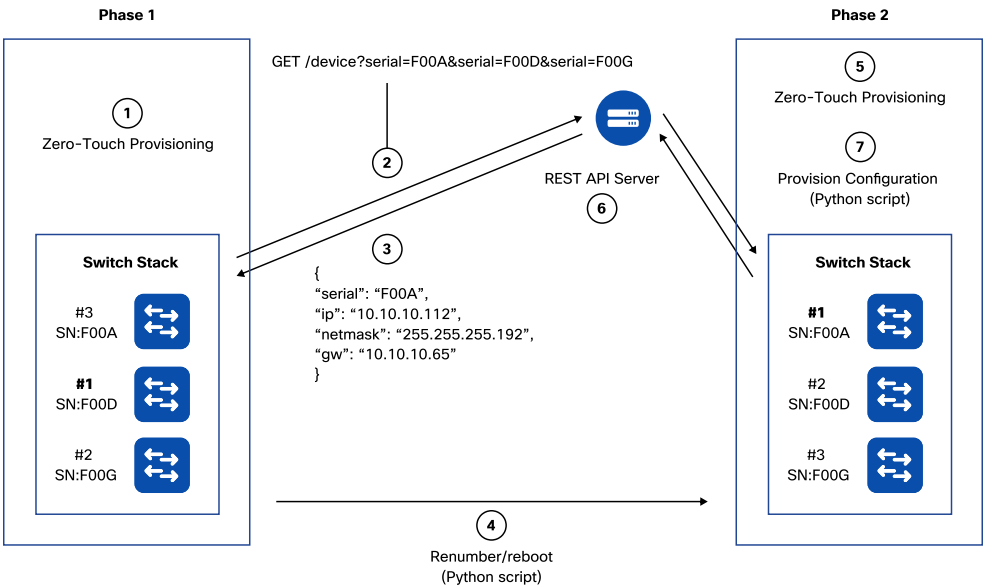
Switch stacking can create challenges for Day 0 deployments as the order in which switch members appear is not deterministic. Ideally, switch numbering should be logically and physically aligned. For example switch 1 is the top of the stack while switch 2 is underneath. The interface configurations on a switch are linked to the logical stack member number. For example, interface Gigabit Ethernet1/0/24 is on logical switch 1. This is particularly important for uplinks as they are not normally spread across every switch in the stack.

The dynamic configuration above can be extended to include a "renumbering" phase. The REST API call includes all of the serial numbers in the stack and returns the serial number of the device that should be the top of the stack, based on a predefined rule. The Python script will then issue a **renumber stack** command if the serial number

returned is not the top of stack switch. As the configuration was not saved, at this point, the stack will be rebooted and the ZTP process will begin again.

The stack is now numbered correctly, and the API will return the same response as before. There is no need to renumber or reboot and the ZTP process completes and saves the configuration file.

DIAGRAM *Advanced ZTP Workflow with Stacking*



The ZTP process can be as simple or as sophisticated as needed. There is complete flexibility of the workflow and the location where state information is obtained and stored.

Considerations

Some aspects to consider when building a ZTP workflow include:

- **Existing configuration:** Ensure that the device is reset to factory defaults by following the device's guideline to restart the ZTP process.
- **Front panel ports:** Check whether the device supports ZTP on the front panel or the management port.
- **Testing/validation:** Extra work is required up front to test and validate the workflow across different deployment scenarios.
- **Wide Area Network (WAN):** Consider the location of images if upgrading over the WAN.
- **Security:** There are considerations around security and device certificate management and for the encrypted communications.

Next Steps

ZTP is a powerful "Day 0" automation tool which can bring up hundreds of networking devices in minutes.

For additional examples, visit *https://github.com/aradford123/ZTP-samples* for sample ZTP Python scripts.

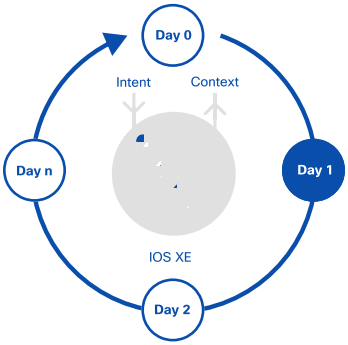
Also refer to:

https://cs.co/ztp_provisioning

https://developer.cisco.com/docs/ios-xe/#day-zero-provisioning-quick-start-guide

https://developer.cisco.com/site/standard-network-devices

YANG



Overview

What is a Data Model?

A data model is a description of how data must be encoded for information exchange between two entities.

A data model can be created for any information exchange process and is based on key-value pairs.

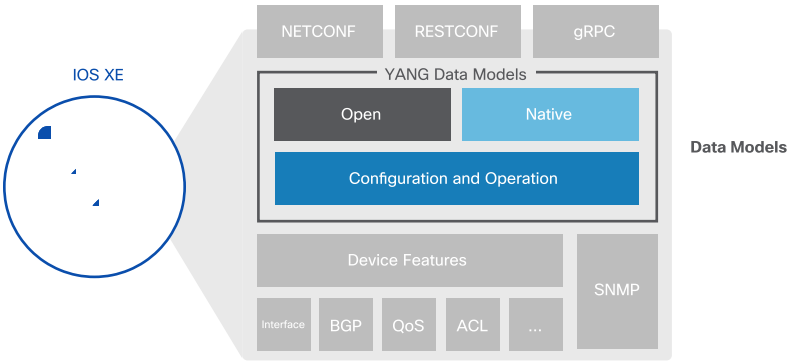
For example, to represent an interface on a switch through a data model, the model might be structured to include:

- **Type**: selection from a pre-defined list (for example Gigabit Ethernet IEEE 802.3z, FastEthernet IEEE 802.3, etc)
- **Description**: free-form text string
- **Status**: Enabled/Disabled (Boolean)

In this example, the bolded items are referred to as "keys" and the "value" follows.

The standard YANG data modeling language imposes a defined structure for generating YANG data models to provide vendor-agnostic approaches.

DIAGRAM *YANG Data Models*

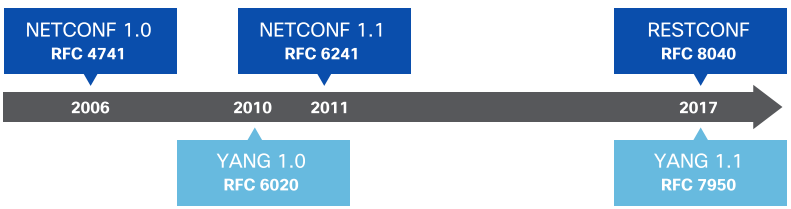


YANG Data Model History

SNMP MIBs and CLIs were not effective as programmatic interfaces. The NETwork CONFiguration Management Protocol (NETCONF), RFC 4741, was standardized in 2006 to address this need. The original NETCONF standard did not specify a data schema, resulting in inconsistency.

The YANG (Yet Another Next Generation) data modeling language and schema were introduced by the IETF as RFC 6020 to address this problem. Subsequently, the NET CONF standard was updated to RFC 6241 to explicitly call out the use of YANG data models.

DIAGRAM *Network Configuration Management Standards Evolution*

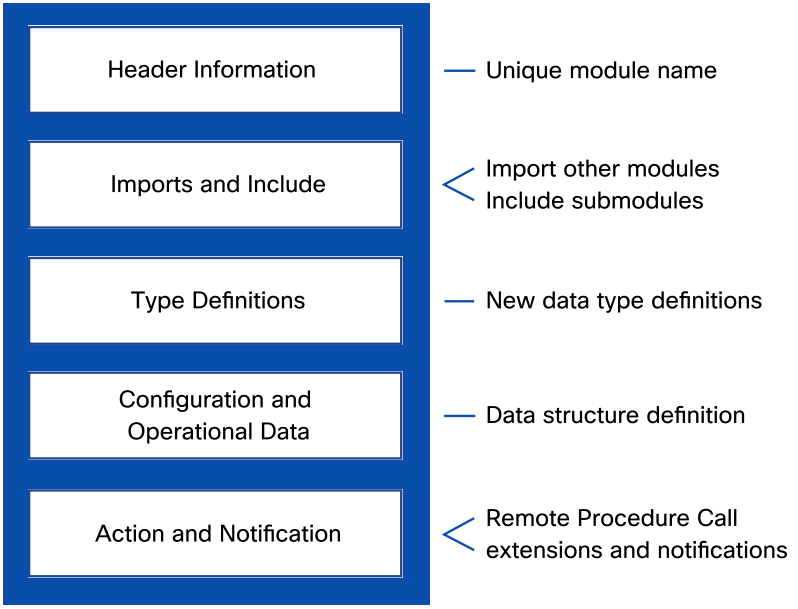


YANG Data Model Summary

YANG is a text-based data modeling language designed for use with any network management protocol including NETCONF and REST Configuration (RESTCONF). It is a modeling language for networking data that was recently updated in RFC 7950 (YANG version 1.1).

YANG data models are composed of modules which represent individual YANG files. The following diagram provides a logical representation of the structure of a YANG module.

DIAGRAM *YANG Data Model Structure*



YANG data models may be used to represent both configuration and operational data, in addition to RPCs and notifications.

The following table describes the differences between MIBs and YANG modules.

TABLE *MIBS vs YANG Data Models*

	MIB	YANG
IETF Standard	Yes	Yes
Transport Protocols	SNMP	NETCONF, RESTCONF, gRPC
Encoding Schema	BER	XML, JSON, GPB
Human Readable	No	Yes
Vendor-Specific Definitions	Yes	Yes
Vendor-Agnostic Definitions	Yes	Yes
Automated Capabilities Exchange	No	Yes
Configuration Management	Rarely	Yes
State and Operational Data	SNMP Get SNMP Traps	Ad hoc queries Dynamic and Configured Telemetry Subscriptions

YANG Concepts

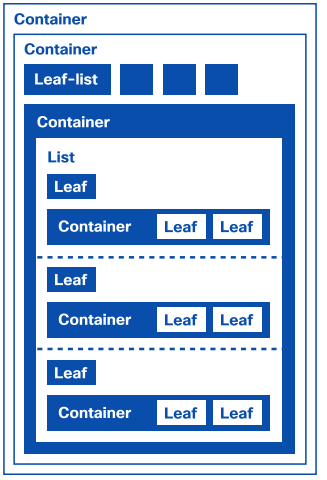
YANG modules are hierarchical tree structures for organizing data. Some of the main concepts are highlighted in this section.

Module Constructs

Modules usually root from one or more containers, and the schema tree extends to another level of data structures. A branch ends with leaves and/or leaf-lists. There are four major statements in a YANG module:

- **Container:** A grouping of other statements (has no "Value")
- **List:** Multiple records containing at least one Leaf "Key" and an arbitrary hierarchy of other statements
- **Leaf:** Single key/value pair
- **Leaf-list:** Multiple key/values pair of the same type

DIAGRAM *Major YANG Constructs*



YANG Data Types

The most commonly used data types supported by YANG data models are:

- **Integer**: signed and unsigned integers, ranging from 8-64bits
- **Decimal64**: decimal number, represented in 64bits
- **String**: human-readable text
- **Boolean**: "true" or "false" value

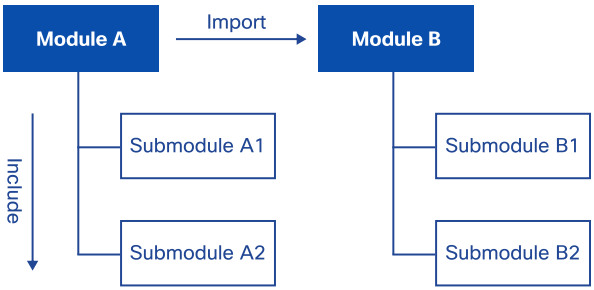
- **Enumeration**: values from a set of assigned names
- **Leafref**: references a particular leaf in the data tree

For network devices, some commonly used data types such as IPv4-address and IPv6-address are defined in RFC 6991 Common YANG data types.

Module Dependencies

Most YANG module files are not self-contained to promote module reusability. Modules can "import" or "include" other modules and sub-modules as needed. The following diagram illustrates a conceptual view of "import" and "include" statements.

DIAGRAM *YANG Import and Include Statements*



Namespaces

A namespace is a mechanism to resolve ambiguity where attributes that have identical names. An example of a namespace is:

```
http://openconfig.net/yang/network-instance
```

Each module has a unique namespace and a unique prefix. Because there are dependencies among modules, a namespace may change in the data of a module.

Module Augmentations and Deviations

When a YANG module is defined by a standards body such as the IETF, IEEE or OpenConfig (OC), vendors have the option to augment or deviate.

- **Augment**: An "augment" statement is the means for a module or a sub-module to add nodes to a schema tree defined in another module.
- **Deviate**: A "deviate" statement defines elements that are different than the standard.

Below is an example of an IOS XE implementation of the OpenConfig Network Instance YANG data model which deviates from the published definition:

```
module cisco-xe-openconfig-network-instance-deviation {
  namespace "http://cisco.com/ns/yang/cisco-xe-openconfig-network-instance-
deviation";
  prefix oc-netinst-devs;
  import openconfig-network-instance {
    prefix oc-netinst;
  }
  import Cisco-IOS-XE-types {
    prefix ios-types;
  }
  ...
  deviation /oc-netinst:network-instances/oc-netinst:network-instance/oc-
netinst:config/oc-netinst:route-distinguisher {
    deviate replace {
      type ios-types:cisco-route-distinguisher;
    }
    description "Replace OC type with Cisco type that supports expanded range of
values";
  }
}
```

The specific container "fdb" in the example above is not supported in the Cisco IOS XE implementation.

Visualization of a YANG Data Model

A YANG module is best understood when visualized as a tree structure. The example below shows the "openconfig-interfaces" module:

```
module: openconfig-interfaces
  +--rw interfaces
    +--rw interface* [name]
      +--rw name                    -> ../config/name
      +--rw config
      |  +--rw name?                    string
      |  +--rw type                     identityref
      |  +--rw mtu?                     uint16
      |  +--rw description?             string
      |  +--rw enabled?                 boolean
      |  +--rw oc-if-cisco:bandwidth
      |     +--rw oc-if-cisco:kilobits?   uint32
      +--ro state
      |  +--ro name?                      string
      |  +--ro type                       identityref
      |  +--ro mtu?                       uint16
      |  +--ro description?               string
      |  +--ro enabled?                   boolean
      |  +--ro ifindex?                   uint32
      |  +--ro admin-status               enumeration
      |  +--ro oper-status                enumeration
      |  +--ro last-change?               oc-types:timeticks64
...
```

YANG Native vs Open Data Models

Cisco IOS XE supports two categories of YANG data models: native and open.

Native Data Models

Cisco IOS XE native data models specific to IOS XE are not interoperable with other platforms. They closely mirror the structure of the IOS XE Command-line Interface (CLI), which makes them more familiar to experienced users of the IOS XE. The key benefit of native data models is the breadth of feature coverage.

Examples include:

- **Cisco-IOS-XE-interfaces.yang:** A native module for configuring interface parameters such as IP address, description, and speed. It also contains operational data about interfaces, such as admin state and InOctets.
- **Cisco-IOS-XE-ospf.yang:** A native module for configuring Open Shortest Path First (OSPF) routing processes.

Open Data Models

Open data models provide a common interface across multiple platforms. Cisco IOS XE supports a number of open data models from both the IETF and OpenConfig standards bodies.

IETF

The Internet Engineering Task Force (IETF) defines many of the standards needed for the operation of networks, including the IP protocol. IETF has released several open data models which are supported by Cisco IOS XE.

Some of the IETF YANG modules supported by IOS XE include:

- **ietf-interfaces.yang**: IETF module for configuring interface parameters such as IP address, description, speed. It also contains operational data about interfaces, such as admin state and InOctets.
- **ietf-ospf.yang**: IETF module for configuring OSPF routing processes.

OpenConfig

OpenConfig (OC) is a consortium of network operators created to define standards intended to make networks more open and programmable. The OpenConfig standards body consists primarily of network operators. As OpenConfig is not a formal standards body, OpenConfig data models change rapidly.

Some of the OpenConfig modules supported by Cisco IOS XE include:

- **openconfig-interfaces.yang**: OpenConfig module for configuring interface parameters such as IP address, description, speed. It also contains operational data about interfaces, such as admin state and InOctets.
- **openconfig-ospfv2.yang**: OpenConfig module for configuring OSPF routing processes

Using Different Model Types

Regardless of which data model is used, the resulting configuration applied to the device is the same. For example, the XML below was generated based on the Cisco IOS XE native data model for configuring interfaces:

```
<native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <interface>
    <GigabitEthernet>
      <name>1/0/24</name>
      <description>Configured by NETCONF!</description>
    </GigabitEthernet>
  </interface>
</native>
```

Note the structure of the data model closely follows the structure of the IOS XE CLI. Now compare the equivalent XML generated from the IETF data model:

```
<interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
  <interface>
    <name>GigabitEthernet 1/0/24</name>
    <description>Configured by NETCONF!</description>
  </interface>
</interfaces>
```

There are some obvious differences between these two models. For example, in the native model, the name of the interface is "1/0/24" whereas in the IETF model the interface is "GigabitEthernet 1/0/24". Despite these differences, IOS XE will render the YANG-modeled data in exactly the same way, as can be seen by examining the CLI after applying the change:

```
switch#show run interface g1/0/24
interface GigabitEthernet 1/0/24
description Configured by NETCONF!
```

The native models provided by Cisco IOS XE will support a far wider variety of features than an open model. Some features are vendor-specific or proprietary, and will not be mod- eled by standards bodies. For example, Enhanced Interior Gateway Routing Protocol (EIGRP) and Cisco Discovery Protocol (CDP) are Cisco features and it is unlikely IETF or OpenConfig would release a model supporting them. Additionally, standards bodies are not likely to support all of the given options under a particular model. For example, while IETF and OpenConfig both have data models covering in terface configuration, they will not support every parameter that is configurable under an IOS XE interface. Open models tend to support the lowest common denominator among vendors.

Standards-based YANG modules, therefore, are the best choice in multi-platform or multi-vendor environments where it would be inconvenient to base configurations on a number of different device models. By using a single and consistent model structure, the same configuration can be easily used on disparate platforms. Conversely, native models are more suited to homogeneous environments which largely consist of a single platform and vendor. An alternative approach is to use open models for more general configuration, but then use native models to tweak specific features.

YANG Data Model Highlights

SNMP MIBs

Cisco IOS XE supports several operational YANG data models which were converted directly from SNMP MIBs. These data models mirror the SNMP MIB exactly and were created to ease migration from SNMP to YANG-based management.

For example, IOS XE supports a MIB for reading Cisco Discovery Protocol (CDP) data via SNMP. This MIB is called CISCO-CDP-MIB. IOS XE also has an equivalent YANG data model called CISCO-CDP-MIB.yang. This data model follows the MIB structure exactly. An example of the YANG tree for a MIB follows:

```
+--ro cdpInterfaceTable
|  +--ro cdpInterfaceEntry* [cdpInterfaceIfIndex]
|     +--ro cdpInterfaceIfIndex            int32
|     +--ro cdpInterfaceEnable?            boolean
|     o--ro cdpInterfaceMessageInterval?   int32
|     +--ro cdpInterfaceGroup?             int32
|     +--ro cdpInterfacePort?              int32
|     +--ro cdpInterfaceName?              snmpv2-tc:DisplayString
```

NETCONF Access Control Model

Data model-based Authentication, Authorization and Accounting (AAA) is based on RFC 6536's NETCONF Access Control Model (NACM). Using data model-based AAA, the user defines different privilege levels and sets rules for each level. Four different types of access control are permitted using data model-based AAA:

- **Protocol operations**: restricts users to only specific RPC or operations. For example, if privilege 5 is limited to get-config operations, and a user logged in at level 5 attempts an edit-config operation, it will fail.
- **Module name**: restricts users to only specific YANG modules. For example, if privilege 5 is limited to the Cisco-IOS-XE-native data model, and a user logged in at level 5 attempts to access the IETF-interfaces module, the operation will fail.

- **Data node**: restricts users to only specific nodes in a YANG tree. For example, if privilege 5 is limited to only the /native/hostname node, and a user logged in at level 5 attempts to write a VRF config, the operation will fail.
- **Notifications**: restricts users to receive only specific notifications.

NACM allows the user to define 15 privilege levels, with 15 being an administrative user. Cisco IOS XE has 15 user privilege levels which correspond to the NACM levels. How ever, the rules defined for NACM are independent of AAA rules for normal user authentication of IOS XE.

A NETCONF user authenticates into the device using standard AAA. The device then forwards the authentication and authorization request to a RADIUS server. Based on the user credentials, the RADIUS server will return the correct authorization level to the device, which will then enforce the rules appropriate to the authentication level of the user.

YANG Relationship to XML and JSON

A common source of confusion is the relationship between YANG data models and encoding formats such as XML/JSON.

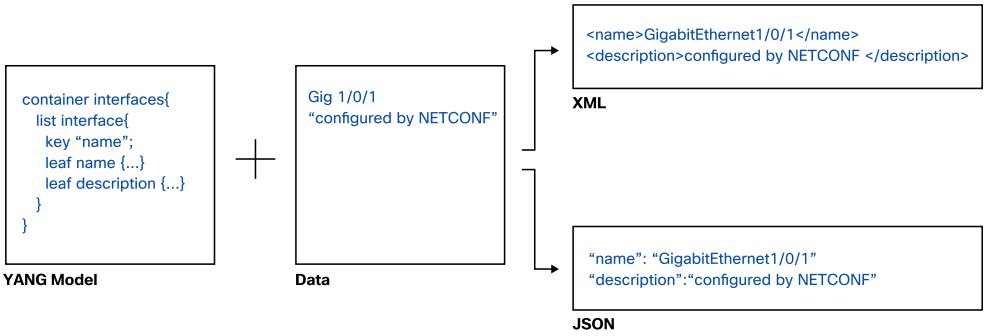
YANG used to represent the data on a device in an abstract way, but does not contain actual device configuration or operational data; it simply shows the structure. In other words, YANG forms the template from which XML/JSON data is generated and does not represent the actual data.

For example, consider a router interface that has an IP address of 10.10.10.10 and description of "Configured by Python". The YANG module for interfaces describes the data that can be associated with an interface. The interface may have an IP address formatted as four octets in dotted decimal notation as well as a description which is a string. The YANG module does not specify the IP address 10.10.10.10 or the description.

When an application reads the configuration of a network device, the device sends an XML-formatted or JSON-formatted version depending on the transport protocol in use.

The structure of the XML or JSON is formatted according to the structure defined in the YANG data model.

DIAGRAM *YANG Relationship to XML and JSON*



YANG Tools

There are many tools available to interact with the YANG modules. Some commonly used tools and their applications are described below.

The YANG Development Kit (YDK) and YDK-Py

The YANG Development Kit (YDK) facilitates network programmability using data models. YDK can generate APIs in a variety of programming languages using YANG modules. These APIs can then be used to simplify the implementation of applications for network automation.

YDK has two main components: an API generator (YDK-gen) and a set of generated APIs. YDK-gen takes YANG modules as input and produces APIs that mirror the structure of the modules. For the generated APIs, Python (YDK-Py), C++ (YDK-Cpp), and Go (YDK-Go) frameworks are available.

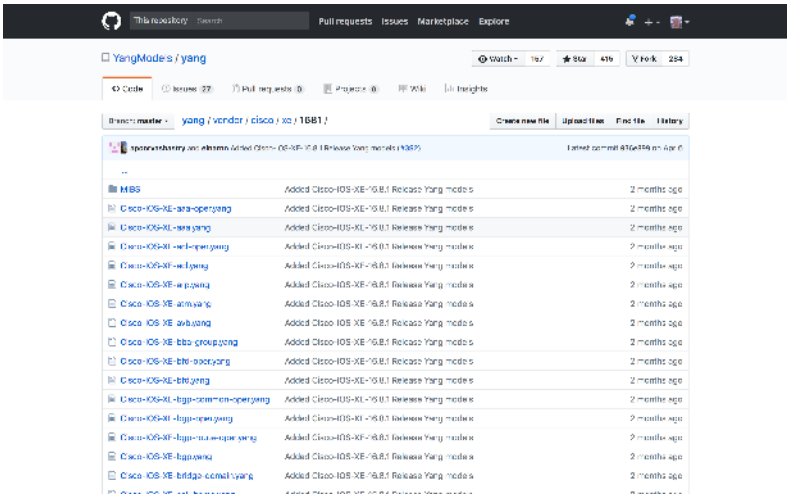
More details on the YDK are available on the Cisco DevNet site at *https://developer.cisco.com/site/ydk/*

GitHub Modules Repository

The YANG modules for Cisco IOS XE are posted to the GitHub site at https://github.com/YangModels/yang/tree/master/vendor/cisco/xe

Modules from multiple versions, products, and vendors are also stored at this location. Modules can also be downloaded or exported from the running device instead of using third-party sites. The illustration below shows the GitHub page with all the YANG native modules included in the IOS XE 16.8.1 release:

DIAGRAM *GitHub Repository Example*

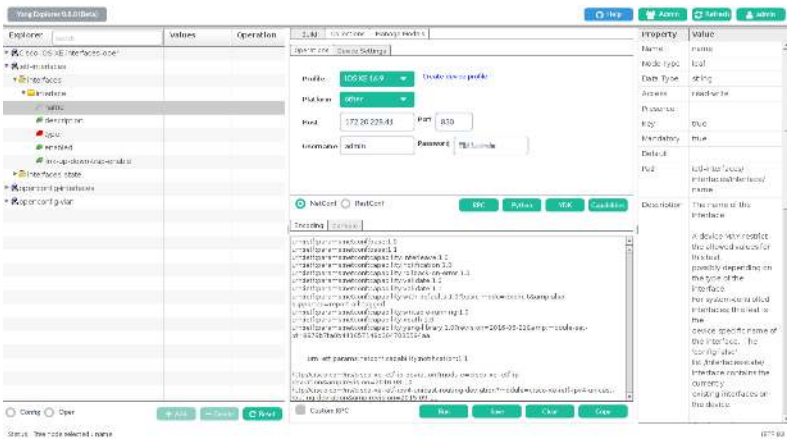


YANG Explorer

YANG Explorer is available on the Cisco DevNet GitHub site at *https://github.com/CiscoDevNet/yang-explorer* and runs as a service on Mac and Linux.

Access the YANG Explorer interface at *http://localhost:8088* Once installed, it can be used to download modules, generate config, and interact with the network device over a NETCONF session.

DIAGRAM *YANG Explorer GUI*



pyang

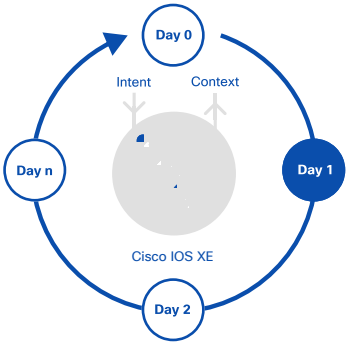
pyang is one of the most common and easily used tools for understanding YANG modules. It produces a tree view showing the structure of a module as well as field names and types. There is a large range of plugins available for advanced users. pyang can be installed by PIP or from the GitHub repository.

```
pyang -f tree openconfig-interfaces.yang
          module: openconfig-interfaces
            +--rw interfaces
               +--rw interface* [name]
                  +--rw name             -> ../config/name
                  +--rw config
                  |  +--rw name?            string
                  |  +--rw type             identityref
                  |  +--rw mtu?             uint16
                  |  +--rw loopback-mode?   boolean
                  |  +--rw description?     string
                  |  +--rw enabled?         boolean
                  +--ro state
                  |  +--ro name?            string
                  |  +--ro type             identityref
                  |  +--ro mtu?             uint16
                  |  +--ro loopback-mode?   boolean
                  |  +--ro description?     string
                  |  +--ro enabled?         boolean
                  |  +--ro ifindex?         uint32
                  |  +--ro admin-status     enumeration
                  |  +--ro oper-status      enumeration
                  |  +--ro last-change?     oc-types:timeticks64
          ...
```

YANG Catalog

The YANG Catalog at https://yangcatalog.org is a module catalog and registry for searching through the large number of published YANG modules. It can be used as a reference to search for what is available from existing modules, contains modules from many vendors, and is not limited to Cisco IOS XE.

Network Device APIs

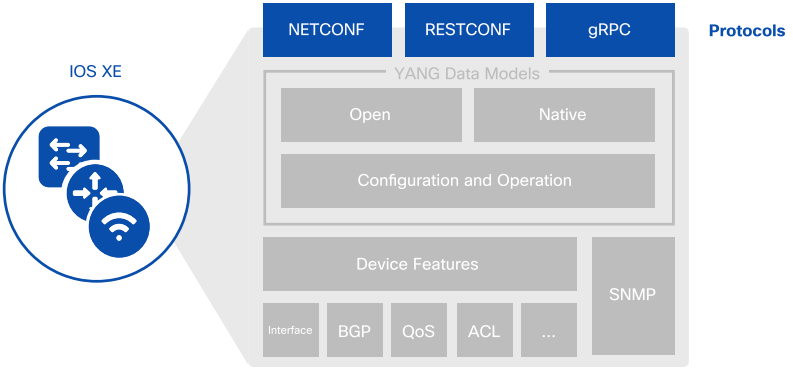


Overview

YANG organizes device configuration and operational states in data models. NETCONF and RESTCONF are standard protocols that allow access to this data.

Data sent using these protocols need to be encoded in a format easy for both sides of the connection to understand. XML and JSON are two commonly used encoding formats. The NETCONF protocol requires data encoded in the XML format. RESTCONF can be encoded in either XML or JSON, although generally JSON is preferred.

DIAGRAM *Programmatic Interface Protocols*



NETCONF

Introduction

Network operators historically used two methods to manage network devices: Simple Network Management Protocol (SNMP) and device-specific Command-line Interface (CLI). Neither is suitable for automated, programmable networks.

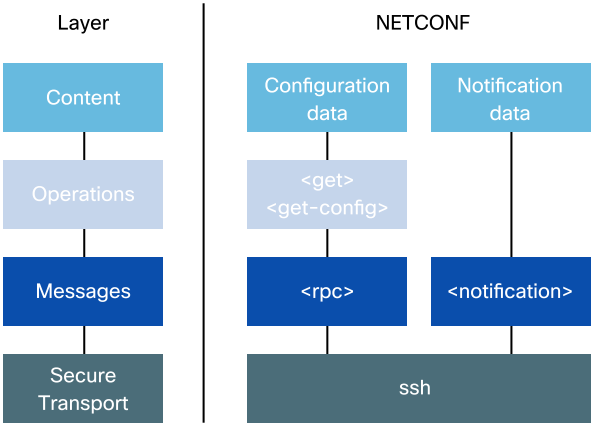
SNMP has a number of limitations, including:

- no discovery process for finding the correct Management Information Base (MIBs)
- no transactional model and rollback
- limited industry support for configuration MIBs

NETCONF stands for NETwork CONFiguration and was originally defined in RFC 4741 for NETCONF 1.0 and updated in RFC 6241 for NETCONF 1.1. The NETCONF protocol defines a set of operations to manipulate structured data defined by YANG data models. It is a Remote Procedure Call (RPC)-based protocol where data is encoded in XML. Transactions are at the core of the protocol stack. It is session- oriented, provides a capabilities exchange for discovering models, and uses a Secure Shell (SSH) as a transport. The combination of NETCONF with YANG constitutes an Application Programming Interface (API).

The following diagram shows the different layers involved when using NETCONF.

DIAGRAM *NETCONF Stack*



NETCONF was designed to enable:

- multiple configuration datastores (candidate, running, startup).
- device-level and network-wide transactions.
- configuration testing and validation.
- a distinction between configuration and operational data.
- selective data retrieval with filtering.
- streaming and playback of event notifications.
- extensible remote procedure calls.

NETCONF has different datastores. These datastores are the target of configuration reads and writes. The datastores defined in the RFC are:

- running (mandatory)
- candidate (optional)
- startup (optional)

SNMP is unable to discover MIBs supported by the device. NETCONF has a built-in capability exchange, providing all supported data models as well as additional operations and datastores supported on a device. For example, "notification" or "roll back-on-error" capabilities are shown below.

```
urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:notification:1.0
urn:ietf:params:netconf:capability:rollback-on-error:1.0
urn:ietf:params:xml:ns:yang:ietf-interfaces?module=ietf-interfaces
http://openconfig.net/yang/interfaces?module=openconfig-interfaces
http://cisco.com/ns/yang/Cisco-IOS-XE-interface-common?module=Cisco-IOS-XE-interface-
common
...
```

Operations

The capabilities exchange shows additional optional operations such as ":validate". All devices support base operations, including:

- **<get-config>**: retrieve all or part of a specified configuration from a named datastore
- **<get></get>**: retrieve running configuration and device state information
- **<edit-config>**: load all or part of a specified configuration to the specified target configuration.
- **<copy-config>**: create or replace an entire configuration datastore with the contents of another complete configuration datastore
- **<delete-config>**: delete a configuration datastore
- **<lock> and <unlock>**: activate a short-lived lock and unlock the configuration system of a device
- **<close-session> and <kill-session>**: gracefully close or force termination of a NETCONF session

The following is an example of a NETCONF RPC operation:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <get-config>
    <source>
      <running/>
    </source>
    <filter xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interfaces/>
    </filter>
  </get-config>
</rpc>
```

These are some of the characteristics of the NETCONF RPC above:

- XML encoding
- <get-config> operation to retrieve configuration
- source data store is :running
- <filter> tag indicates specific data of interest

Cisco IOS XE Implementation

Cisco IOS XE supports two datastores, running and candidate. It also supports locking of the datastores, as well as con guration rollback.

Datastores

The running datastore is the default and the candidate is optional. To enable the candidate datastore, the following command is used: `netconf-yang feature candidate-data-store`. When the candidate datastore is enabled, some processes in IOS XE are restarted, and any existing NETCONF sessions are terminated. "urn:ietf:params:net-conf:capability:candidate:1.0" is advertised in the server capabilities for new NETCONF sessions. NETCONF clients can make changes to the conguration in the candidate datastore, and validate the data before it is written to the running datastore using the RPC. This provides transactionality.

Datastore Lock and Validation

The best practice is to lock the running datastore before locking the candidate datastore, to prevent a change to the running-config before the candidate is committed. Send the "validate" RPC to validate the candidate datastore before commit.

Without a candidate datastore support, the running datastore can be modified directly using the following sequence:

1. lock running datastore
2. send edit-config to running-config
3. unlock running-config

Configuration Rollback

The NETCONF protocol supports rollback capability that automatically preserves the state of a device in case of configuration failures.

High Availability

The redundant hardware and software in a device are transparent and appear as a single entity. The NETCONF clients only modify the active configuration which is synced automatically to the standby.

When a failover occurs, NETCONF clients will have to re-establish their sessions to the device. Any configured subscription is maintained across sessions. Dynamic subscriptions are tied to NETCONF sessions and are lost when a session is closed.

RESTCONF

Introduction

RESTCONF provides a standard REST-based programmatic interface to access network devices for configuration and operation. RESTCONF is an HTTP-based protocol that uses structured data in XML or JSON format.

REST APIs

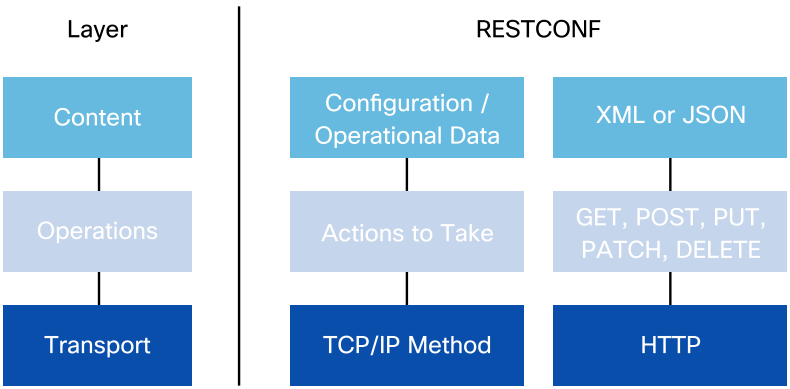
RESTCONF emerged in response to the broad industry adoption of REST APIs. REST APIs are scalable, provide a simple and uniform interface, and are easy to port to different platforms. REST APIs share the following properties:

- client-server model
- does not maintain per-client state
- textual representation with XML or JSON
- resources specified in Uniform Resource Identifiers (URIs)
- pre-defined HTTP verbs e.g. GET, POST, etc

RESTCONF provides a standard REST API based on YANG data models.

Stack

DIAGRAM *RESTCONF Stack*



Methods

The RESTCONF protocol uses HTTP methods to identify CRUD (Create, Read, Update and Delete) operations for a particular resource. The most common operations include:

- **POST**: "create" a data resource or invoke an operation resource
- **GET**: "read" data and metadata for a resource
- **PUT**: "update" the target data resource
- **DELETE**: "delete" a data resource

The following is an example of a RESTCONF GET operation to retrieve the interface configuration from a device

```
curl -X GET -u admin  https://10.10.10.112/restconf/data/ietf-interfaces:interfaces"-
interfaces:interfaces
  -H 'Accept: application/yang-data+json'
  -H 'Cache-Control: no-cache'
```

Cisco IOS XE Implementation

The Cisco IOS XE implementation supports both running and candidate datastores. Similar to NETCONF, RESTCONF supports editing the candidate datastore and committing the changes.

Datastores

RESTCONF clients can write to the candidate datastore if enabled by CLI, as explained in the NETCONF section. The RESTCONF request is followed implicitly by a commit in the same transaction. If the candidate datastore is not enabled by CLI, all RESTCONF requests go to the running datastore directly.

Configuration Lock and Rollback

RESTCONF does not have lock capability, but the transactional behavior is guaranteed within one request. Since each RESTCONF request is a transaction, any error in it will cause a config rollback in Cisco IOS XE.

Comparison of NETCONF and RESTCONF

At a high level, NETCONF is more comprehensive, flexible, and complex than RESTCONF. RESTCONF is easier to learn and use for engineers with previous REST API experience. The following are differences between NETCONF and RESTCONF:

- NETCONF supports running and candidate datastores, while RESTCONF practically supports only the running datastore as any edits of candidate datastore are immediately committed.
- RESTCONF does not support obtaining or releasing a datastore lock. If a datastore has an active lock, the RESTCONF edit operation will fail.
- A RESTCONF edit is a transaction limited to a single RESTCONF call.
- RESTCONF does not support transactions across multiple devices
- Validation is implicit in every RESTCONF editing operation which either succeeds or fails.

The following table shows the different NETCONF operations and how they map to RESTCONF methods:

TABLE *NETCONF Operations and RESTCONF Methods*

DESCRIPTION	NETCONF	RESTCONF
Create a data resource	`<edit-config> </edit-config>`	POST
Retrieve data and meta-data	`<get-config>, <get> </get-config>`	GET
Create or replace a data resource	`<edit-config> (nc:operation="create/replace")`	PUT
Delete a data resource	`<edit-config> (nc:operation="delete");`	DELETE

Next Steps

DevNet Learning Labs

The Cisco DevNet site at *https://developer.cisco.com* includes learning labs, sandboxes, coding examples and other resources for learning about RESTCONF and NETCONF.

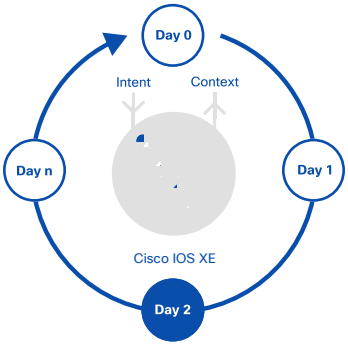
Tools

ncclient is a Python library provided for NETCONF and can be installed with PIP.

netconf-console is based on ncclient and can be installed with PIP.

Postman is for REST API testing and is available from *https://www.getpostman.com/*

Telemetry



Overview

Telemetry is defined as the automated communications process by which measurements and other data are collected at remote locations and transmitted to receiving equipment for monitoring. The word is derived from the Greek roots: *tele* = remote, and *metron* = measure. In the context of Cisco IOS XE, the networking devices are the remote devices which transmit operational, configuration, event and flow data to data collectors.

Common telemetry use cases include business intelligence for network, application, and user monitoring, network planning, security analysis, accounting, logging and traffic engineering.

Operational Data

Operational data represented in YANG data models can be streamed as telemetry to remote collectors for monitoring and processing. Subscriber applications, via the ietf-yangpush YANG data model, request continuous and customized streams of up dates from a Cisco IOS XE device. The streaming telemetry notifications are based on the data model supported by the device which makes possible near real-time operational and configuration insights. There are two subscription types and two notification types described below.

Subscription Types

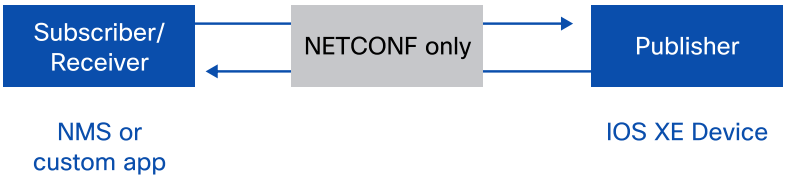
The application specifies the type of subscription to established, either dynamic or configured.

Dynamic

The subscriber sends a request via the ietf-yangpush.yang data model. If the Cisco IOS XE device approves the request, it replies with a subscription-id and starts streaming telemetry data. Dynamic subscriptions cannot be modified but can be terminated at any time. Dynamic subscriptions automatically terminate if the NETCONF session is terminated.

In a dynamic subscription, the subscriber and receiver are always the same entity as represented in the diagram below.

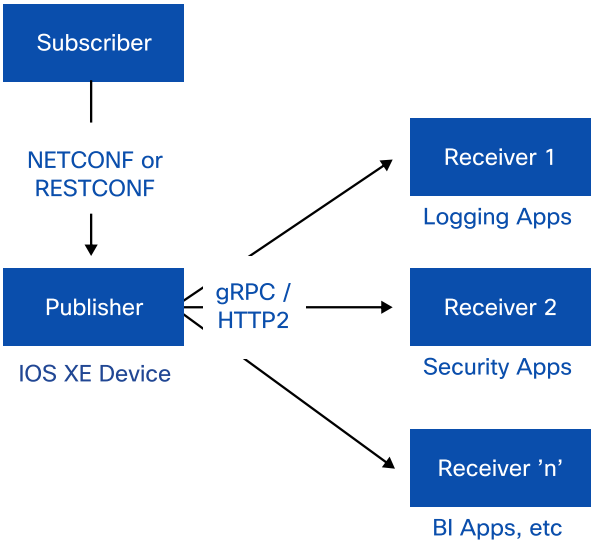
DIAGRAM *Dynamic Subscription*



Configured

In the configured subscription, the receiver is not necessarily the subscriber and the subscription is configured via CLI, NETCONF or RESTCONF. Subscriptions can be modified or terminated at any point in their lifetime. Configured subscriptions are persistent between IOS XE device reboots, unlike dynamic subscriptions. Configured subscriptions can be used to stream data to more than one receiver, as reflected in the diagram below.

DIAGRAM *Configured Subscription*



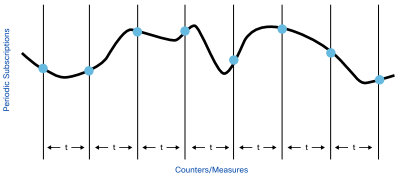
Notification Types

The application can specify how notifications are published to the receivers: either periodically or on-change.

Periodic Notifications

With a periodic notification, subscribed data is streamed according to a fixed cadence defined within the telemetry subscription. Periodic publication is ideal for data which represents device counters or measures such as interface statistics or CPU utilization because they constantly change.

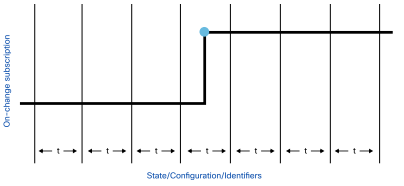
DIAGRAM *Periodic Notifications*



On-Change Notifications

On-change notifications stream data only when data changes. Examples include a configuration change, a fault, crossing a threshold, or the detection of a new neighbor.

DIAGRAM *On-Change Notifications*



Flow Data

NetFlow is an integral part of Cisco IOS XE software. It measures data as it enters specific interfaces and forwards metadata to a receiver. NetFlow allows extremely granular and accurate traffic measurements and high-level aggregated traffic collection. Network Management Systems (NMS) use NetFlow to identify network congestion, collect appli-cation Class of Service (CoS), IP traffic source address, destination addresses, byte counts and more.

DIAGRAM *Stealthwatch Dashboard*



Use Cases

Network Application and User Monitoring

Telemetry data provides detailed, time-based and application-based usage of a network. This information assists planning and allocation of network and application resources, including extensive near real-time network monitoring capabilities. It can be used to display traffic patterns and application-based views. Telemetry enables proactive problem detection, efficient troubleshooting, and rapid problem resolution. This information is used to efficiently allocate network resources and to detect and resolve potential security and policy violations.

Network Planning

Telemetry can be used to capture data over a long period of time to track and anticipate network growth and plan upgrades to increase the number of routing devices, ports, or higher-bandwidth interfaces. Telemetry data helps network planning, which includes peering, backbone upgrade planning, and routing policy planning. It can be used to minimize the total cost of network operations while maximizing network performance, capacity, and reliability. Telemetry data allows the user to identify traffic, validate bandwidth and Quality of Service (QoS), and assist the analysis of new network applications.

Security Analysis

Telemetry data can be used to identify and classify Denial of Service (DoS) attacks, malware, and threats in real-time. Changes in network behavior indicate anomalies that can be detected using telemetry data. The data is also a valuable forensic tool to understand the history of security incidents.

Traffic Engineering

Telemetry can be used to measure the amount of traffic crossing peering or transit points to determine whether the peering arrangement with service providers is fair and equitable.

Subscription Tools

NCC is a set of tools to interact with network devices via NETCONF and is available from *https://github.com/CiscoDevNet/ncc* One of its tools allows the creation of NETCONF notifications. The following example shows the creation of a dynamic, periodic subscription:

```
python ./ncc-establish-subscription.py --host csr1000v --period 3000 --xpaths
"/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds"
```

This will set up a notification on the device csr1000v. It will return the data contained in "/process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds" (the 5 second CPU utilization of the device). The data will be provided every 3000 centiseconds (30 seconds).

This utility creates a callback (a function that processes the notification) to handle the streaming data. Sample output appears below:

```
(Default Callback)
Event time      : 2018-05-08 21:57:53.940000+00:00
Subscription Id : 2147483649
Type            : 1
Data            :
<datastore-contents-xml xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push">
  <cpu-usage xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-process-cpu-oper">
    <cpu-utilization>
      <five-seconds>35</five-seconds>
    </cpu-utilization>
  </cpu-usage>
</datastore-contents-xml>
```

In the example above, the CPU utilization over 5 seconds is shown as 35 percent.

Data Collectors

Network devices stream continuous data to data collectors and provide near real-time access to operational statistics. Below are examples of data collectors.

Cisco Stealthwatch

Cisco Stealthwatch consumes NetFlow data and is optimized for security use cases. By providing Encrypted Traffic Analytics (ETA) capabilities, it provides visibility into encrypted flows between hosts. Use cases include malicious traffic detection, unauthorized application usage, and data exfiltration.

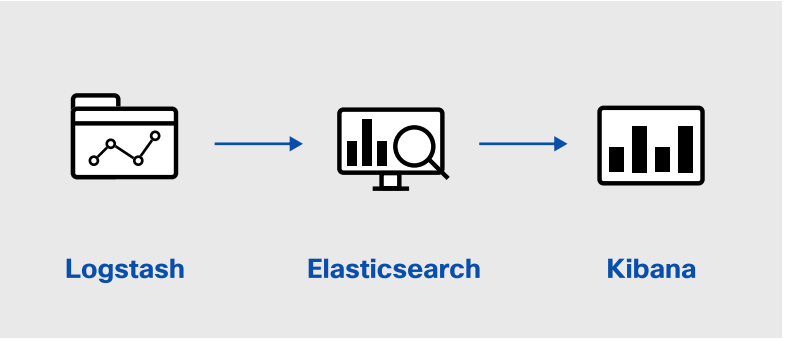
ELK Stack

The ELK stack is a collection of open source applications used for data collection and visualization. The three applications that comprise the stack are:

- **Elasticsearch**: search and analytics engine
- **Logstash:** log data processor
- **Kibana:** visualization frontend

Model-Driven Telemetry is receiving data on the collector in near real-time. More information on the ELK stack is available from the Elastic website at *https://www.elastic.co/elk-stack* The following blog shows how to ingest NetFlow into ELK *https://blogs.cisco.com/security/step-by-step-setup-of-elk-for-netflow-analytics.*

DIAGRAM *ELK Stack Components*

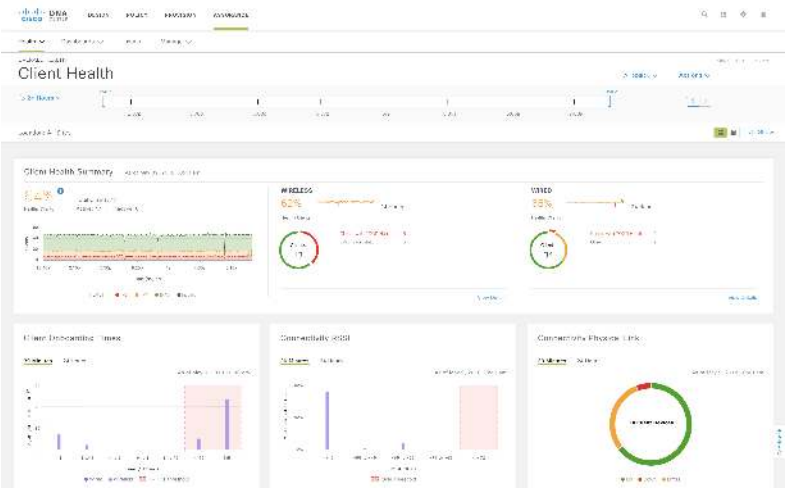


DNA Center

Cisco DNA Center Assurance provides the capabilities for ongoing and proactive monitoring of network status and for alerting network failures. DNA Center uses various methods to communicate with network elements for ongoing monitoring, such as sys-log, SNMP, and streaming telemetry. The DNA Center Assurance solution provides ongoing analysis of monitored Key Performance Indicators (KPIs) and includes the ability to provide trend line analysis.

Additionally, it provides the capability for on-demand troubleshooting by recognizing faults in the network and suggesting recommended actions and by leveraging the Path-trace application, which visualizes the route the data takes on the network. More details about DNA Center Assurance are available at https://cisco.com/go/dna-analytics-and-assurance

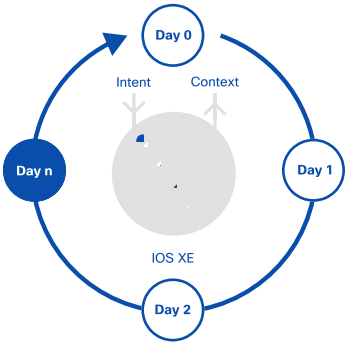
DIAGRAM *DNA Center Assurance Dashboard*



Pipeline

Pipeline is a set of applications that provides scalable data collection with a multi-function collection service. Pipeline receives telemetry data and converts it to a JSON-encoded text file. More details about Pipeline: *https://github.com/cisco/bigmuddy-network-telemetry-pipeline*

Python



Overview

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It uses a simple, easy-to-learn syntax which emphasizes readability. It supports modules and packages and encourages program modularity and code reuse. Python interpreters and standard libraries are available in source or binary form without charge for all major platforms. Additional libraries are widely available throughout the open source community.

Python Libraries

Python libraries are collections of Python modules that allow actions to be performed without having to create new code. Python modules contain definitions and statements and are identified by the ".py" suffix. The standard Python libraries available with Cisco IOS XE running on Catalyst 9000 platforms are based on the library that ships with CentOS 7. Additional Python modules are available through the open source community to further simplify coding efforts.

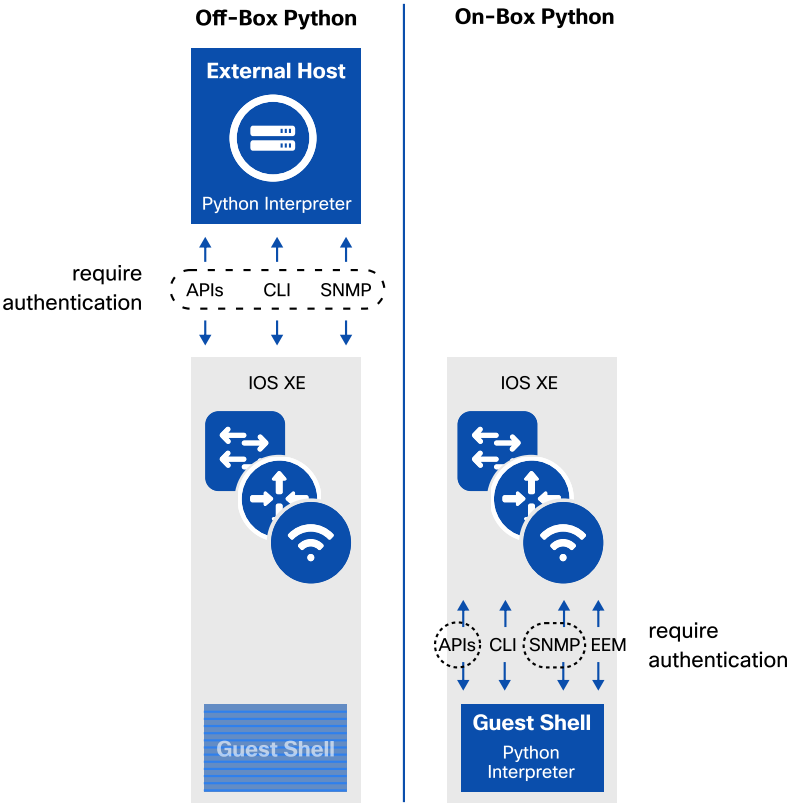
TABLE *Key Python Libraries for Network Programmabilty*

Libary	Description	Source
cli	Executes IOS XE CLI commands	Pre-installed IOS XE specific library, on-box only
eem	Registers Python scripts as EEM policies	Pre-installed IOS XE specific library, on-box only
json	Lightweight data exchange format inspired by inspired by JavaScript object literal syntax	Pre-installed standard library
time	Time access and conversions for various time-related functions	Pre-installed standard library
requests	Allows HTTP/1.1 requests to be sent for use with RESTCONF	pip install requests
ncclient	NETCONF client facilitates client-side scripting and application development for use with NET-CONF	pip install ncclient
netaddr	Represents and manipulates IP addresses	pip install netaddr

On-Box vs Off-Box Python

Cisco IOS XE supports both On-Box and Off-Box Python. On-Box refers to the location of the Python Interpreter which is running within the IOS XE Guest Shell. Off-Box represents an externally hosted Python Interpreter. Using an Off-Box Python Interpreter requires authentication to access the device, while On-Box Python removes this burden as Guest Shell is pre-authenticated.

DIAGRAM *On-Box vs Off-Box Python*



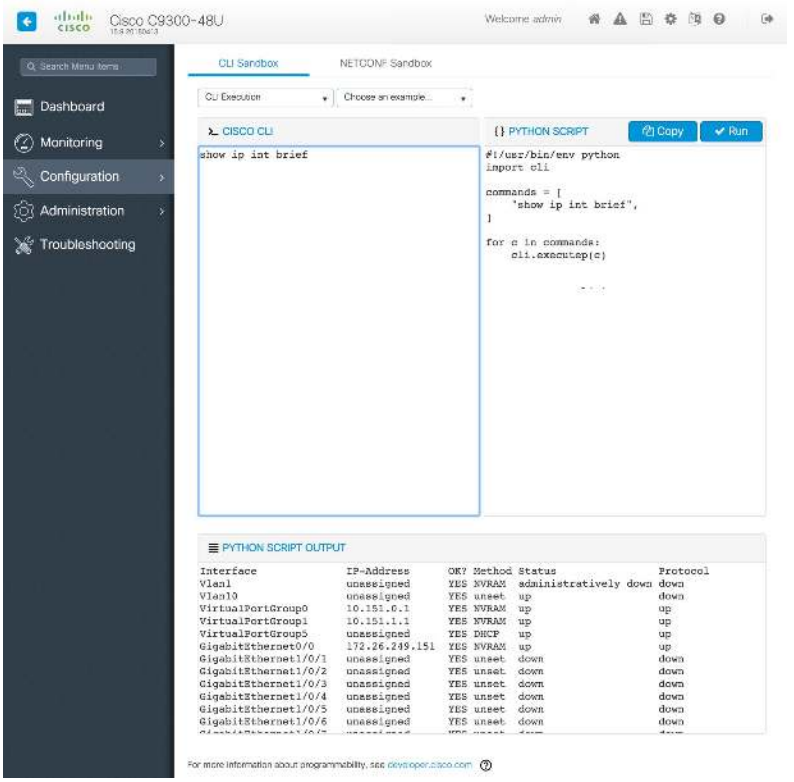
Python WebUI Sandbox

The WebUI Sandbox is a web application that generates Python code from CLI commands or with NETCONF RPCs. The code can be executed, copied, and reused in on-box or off-box scripts throughout the network.

WebUI Sandbox requires:

- Guest Shell to be enabled
- NETCONF to be enabled
- HTTP Server to be enabled
- The ncclient module to be installed within the Guest Shell

DIAGRAM *Python WebUI Sandbox*



On-Box Python

Executing Python code directly on the device is referred to as On-Box Python. On-Box Python can be executed interactively or scripts can be run within the Guest Shell. A Guest Shell container is a built-in Linux Container (LXC) running on Cisco IOS XE with Python version 2 pre-installed.

The Python interpreter within Guest Shell can be run in interactive mode which takes commands and runs them as they are entered.

Additional Python libraries such as "requests" and "ncclient" can be installed and upgraded using the PIP tool. To upgrade Python to version 3, use one of the methods below:

- A stand-alone and self-contained installation tarball when Guest Shell does not have Internet connectivity.
- "yum install" when Guest Shell has network connectivity to the online package repository.

Why On-Box Python?

The following are some use cases for On-Box Python:

- **Zero-Touch Provisioning (ZTP)**: During the ZTP process, a Python script is downloaded to the IOS XE devices and executed within the Guest Shell. This script completes the initial configuration of the device.
- **Embedded Event Manager (EEM)**: A Python script can be executed in response to an event detected by Embedded Event Manager (EEM). For example, if a critical interface goes down, then a Python script can be executed to alert the administrator or return the interface to an operational state.
- **Local Execution**: Python scripts executed in the secured Guest Shell environment or directly from the IOS command line in order to automate tasks.

Off-Box Python is historically the predominant method for Python execution due to its centralized execution model, but there are some very good use cases for on-box as seen above.

Advanced On-Box Python

Executing a Python script from Cisco IOS XE

The simplest way to execute a Python script is via the IOS CLI which will run the script within the Guest Shell environment. Optional parameters that are required by the script can be sent.

```
guestshell run python script.py
```

CLI Custom Python Library

Python scripts running within Guest Shell can use the built-in CLI module to run exec commands and make con g uration changes. This module does not require authentication as the Guest Shell Python interpreter can only be run by a privilege level 15 user.

The following two examples illustrate some functions provided by this module. The first example shows how to run an IOS exec command that prints the response from the IOS show command.

```
#!/usr/bin/env python
from cli import execute
print execute('show clock')
```

The output appears below

```
[guestshell@guestshell ~]$ ./exec.py
04:43:21.179 AEST Wed May 9 2018
```

The second example shows how to change the configuration of the device.

```
#!/usr/bin/env python
from cli import configure
print configure(['int loop12','description created by python'])
```

This script configures a loopback interface with a description. Multiple commands need to be separated by commas. The output shows the results of the commands that were executed.

```
[guestshell@guestshell ~]$ ./configure-interface.py
[ConfigResult(success=True, command='int loop12', line=1, output='', notes=None),
ConfigResult(success=True, command='description created by python', line=2,
output='', notes=None)]
```

The CLI module requires no authentication as it has a private connection to the IOS via HTTP. The **ip http server** must be enabled for this to function.

EEM Custom Python Library

The Embedded Event Manager (EEM) Python library interacts with the EEM infrastructure on the IOS XE device. These functions are only available from a script that has been registered with EEM. There are three modes in which EEM can execute a Python script.

1. Cron Mode

Cron mode executes a script according to the specified schedule. For example, to execute the "monday_15h30m_script.py" script every Monday at 3:30PM, the following IOS XE configuration can be used:

```
event manager applet MondayMorningPython
event timer cron cron-entry "30 15 * * 1"
action 1.0 cli command "enable"
action 1.1 cli command "guestshell run python
bootflash:gs_script/monday_15h30m_script.py"
action 1.2 cli command "exit"
```

2. Countdown Timer Mode

Countdown timer mode executes a script once the timer counts down to zero. In this scenario, the EEM countdown timer is used to execute the Python script "run_in_60_seconds.py" at some time in the future, for example in 60 seconds. For example, IOS configuration follows:

```
event manager applet CountdownPython
event timer countdown time 60
action 1.0 cli command "enable"
action 1.1 cli command "guestshell run python
bootflash:gs_script/run_in_60_seconds.py"
action 1.2 cli command "exit"
```

3. Policy Mode

The Python script is registered natively with the EEM infrastructure and is executed when a specific event occurs. The following line must be at the start of the Python script as it registers the Python script with EEM. For example, EEM executes the Python script when a syslog message with "CONFIG_I" is matched.

```
::cisco::eem::event_register_syslog pattern "CONFIG_I" maxrun 60
```

Once the above line has been added to the Python script, the following commands instruct IOS to look for and register policy scripts in the specified directory. IOS will parse the script and register a trigger.

```
event manager directory user policy flash:
event manager policy config_check.py
```

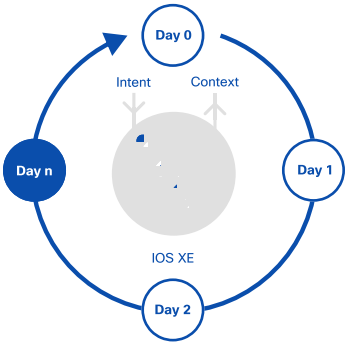
When a configuration change occurs, a syslog entry with the pattern "CONFIG_I" is matched and triggers the script.

Common Issues

These are some common issues encountered while working with Python:

- The correct version must be verified since Python version 2 and Python version 3 are not mutually compatible.
- Python version 3 can be installed on a device that already has Python version 2. Use the **guestshell run python3** command for executing Python version 3 scripts and use **guestshell run python2** or **guestshell run python** for Python version 2 scripts.
- Networking connectivity, proxy, and Domain Name System (DNS) must be configured before downloading additional packages from PIP.
- The required Python packages must be installed on the device.

Guest Shell



Introduction

Guest Shell is an execution space running within a Linux Container (LXC), designed to run Linux applications including Python. It also supports Day 0 device onboarding. The Guest Shell environment is intended for tools, Linux utilities, and manageability rather than network routing and switching functions.

Guest Shell shares the kernel with the host Cisco IOS XE system. While users can access the Linux shell and update scripts and software packages in the container root file system, users within the Guest Shell cannot modify the host file system or processes. Decoupling the execution space from the native host system allows customization of the Linux environment to suit the needs of the applications without impacting the host system.

The Guest Shell container is managed using IOx, which is Cisco's Application Hosting Infrastructure for IOS XE devices. IOx enables hosting of applications and services developed by Cisco, partners, and third-party developers in network edge devices, seamlessly across diverse and disparate hardware platforms.

Security

Guest Shell supports several security features, including:

- The Guest Shell OS binaries and libraries are regularly updated for security compliance.
- The Guest Shell Linux console can only be accessed once the user has authenticated within the IOS XE CLI or WebUI Sandbox.
- The Guest Shell listens on port 22 on the localhost interface and cannot be accessed outside of the IOS XE device.

Configuration and Updates

IOx must be configured and running for Guest Shell configuration to be successful. Configure IOx with the command **iox** in configuration mode.

After IOx services are running, enable Guest Shell with the command **guestshell enable** in exec mode. This command is also used to reactivate Guest Shell, if it has been disabled.

Other useful commands are:

- **guestshell run** *linux-executable* – for example **guestshell run python** or **guestshell run bash**
- **guestshell disable** – shuts down and disables Guest Shell
- **guestshell destroy** – removes the root file system from the flash file system. All files, data, installed applications and custom Python tools and utilities are removed and are not recoverable.

Guest Shell network access is provided by either the management interface or from a Virtual Port Group (VPG) if no management interface is available.

Additional configuration may be required to enable full networking connectivity:

- Configure Domain Name System (DNS) by updating the "/etc/resolv.conf" file
- Configure proxy settings by updating the "/bootflash/proxy_vars.sh" file

The Guest Shell environment comes pre-installed with Python 2.7, Cisco's Embedded Event Manager, and the Cisco IOS XE CLI Python libraries.

Additional libraries can be installed or updated using **pip install** or **pip update.** Other Linux applications can be installed using the Yum, RPM, or Git command-line utilities.

A DNA Network-Essentials license is required to enable Guest Shell capabilities on Cisco IOS XE network devices.

Resource Allocation

Each Cisco IOS XE device has hardware resources available for Guest Shell. These specifica-tions depend on the available hardware. Cisco IOS XE running on a Catalyst 9000 switch reserves dedicated memory and CPU resources for Guest Shell while it doesn't reserve any storage which is shared with all the other IOS XE processes. Guest Shell uses internal flash by default but will use external Solid State Drive (SSD) storage if available.

TABLE *Catalyst 9000 Guest Shell Hardware Resources*

Platform	Total Memory (GB)	Total CPU Cores	Core Speed (GHz)	Guest Shell			
				Memory (MB)	vCPU	CPU units	Bootflash (GB)
Catalyst 9300	8	4	1.8	256	2	800	1.1
Catalyst 9400	16	4	2.4	256	2	800	1.1
Catalyst 9500	16	4	2.4	256	2	800	1.1
Catalyst 9500 high-performance	16	4	2.4	256	2	800	1.1

The following are key characteristics of resource utilization for application hosting and Guest Shell:

- The maximum amount of space is 1.1GB unless external storage is used
- Guest Shell and Application Hosting share CPU and Memory
- Guest Shell uses flash but will use external storage if available

High Availability

High Availability is supported for stacked switches. When Guest Shell is installed, the "/bootflash/gs_script" directory is created in the flash filesystem and contents of this directory are synchronized across all stack members in a switchover event. During a switchover, the new active device creates its own Guest Shell installation and the previous filesystem is not maintained.

Use Cases

EEM Integration

When an Embedded Event Manager (EEM) event is triggered, one of the possible actions is to execute a Python script in the Guest Shell environment.

For more details refer to the Python chapter, or visit: *https://cs.co/ccm_gs*

Zero-Touch Provisioning Integration

When a Cisco IOS XE device boots without a configuration file, it will enter Zero-Touch Provisioning (ZTP) mode. It then obtains an IP address from DHCP along with the URL of the HTTP or TFTP server so it can download and run a Python script from the server. Guest Shell provides the environment for the Python script to run. The script can apply the initial configuration to the device.

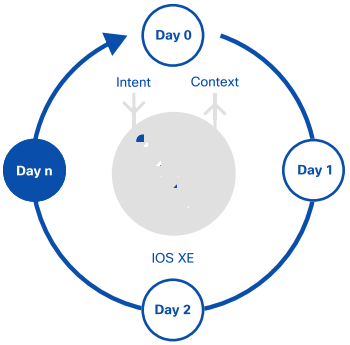
For more details refer to the Python chapter, or visit *https://cs.co/ZTP*

Next Steps

Guest Shell configuration information can be found in the Cisco IOS XE Configuration Guide main page at *https://cs.co/guestshell_config_guide*

More information and practical examples can also be found at the Cisco IOS XE page on DevNet: *https://developer.cisco.com/site/ios-xe/*

Application Hosting



Introduction

Application Hosting makes it possible for network operators to run Linux applications directly on a Cisco IOS XE device. Hosted applications can be developed by partners or third-party developers. The Cisco Application Framework (CAF) manages applications in a containerized environment at the edge.

Application Hosting enables edge computing on networking devices. Edge (or fog) computing is the extension of cloud computing to the edge of an enterprise net work. It changes the way data is processed in comparison to cloud-hosted applications.

Application Types and Use Cases

In the past, network devices and network operating systems were designed and built to execute only specific tasks such as routing and switching traffic rapidly.

The Cisco IOS XE Operating System (OS) is designed to be flexible, both in the hardware and software architecture. The new hardware features, such as multi-core Intel x86 CPUs, increased memory and storage footprints, combined with the Linux-based IOS XE OS, makes it possible for network devices to be used in fog computing. As mentioned above, this opens up completely new use cases such as hosting containers and third-party applications such as network performance monitoring.

Application Hosting Types — VM/LXC/Docker

Cisco IOS XE applications are pre-packaged in containers using different formats:

- **KVM-based Virtual Machines:** Includes application, binaries and libraries, and entire guest OS
- **LXC Linux containers:** Enables OS-level virtualization for multiple Linux systems on a single host

- **Docker containers:** Makes the process of creating and maintaining Linux containers easier

Applications are loaded to the edge devices by use of an application orchestrator. This manages the entire application lifecycle, from installation, activation and upgrade, to uninstallation.

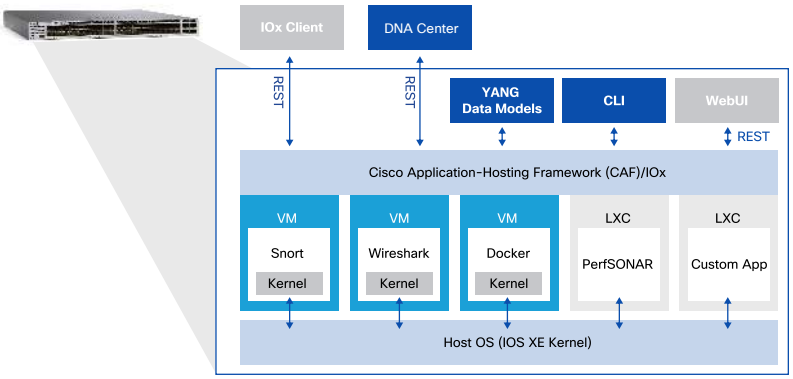
Cisco IOS XE Application Hosting is important for the adoption of new use cases such as security agents, performance monitoring, troubleshooting tools, cloud gateways and custom Python applications.

Cisco Application-Hosting Framework

Cisco has built a new framework to manage applications and hardware resources on network devices: the Cisco Application-Hosting Framework, also known as Cisco IOx. The IOx framework was created to host applications for Internet of Things (IoT) network devices where thousands of devices run small applications. Cisco is now also leveraging the same framework for enterprise IOS XE-based network platforms.

CAF on Cisco IOS XE supports both KVM-based VMs and LXC Linux containers. While native Docker containers are not yet supported, Docker tools can be used to easily build CAF applications in LXC format. A reference KVM-based VM with Docker pre-installed will also be provided.

DIAGRAM *Cisco Application Framework*



Anatomy of a CAF Application

A CAF application is packaged in a standard Linux tar archive format containing several files. One of the most important files in the archive is the IOx package application descriptor file, which contains the following:

- application information (name, description, version, author)
- application type (LXC or VM)
- hardware resources required (CPU, memory, storage)
- container virtual network interfaces
- disk image files (in qcow2 or other format)
- startup tasks

Application Hosting Security

Applications run in either a virtual machine or a Linux container and are isolated from the main operating system. There is isolation between the hosted application and the network device: isolated user space, fault, and resource isolation.

Moreover, only a portion of the system resources (RAM, CPU, and storage) are allocated for a given application, and the hosted application will not be able to consume additional resources and impact on the operations of the IOS XE device.

Finally, applications have no access to the internal device flash storage, which is reserved for IOS XE for integrity reasons.

Hardware Resources

Each Cisco IOS XE network device includes specifications for the hardware resources available for application hosting. These specifications are dependent on the hardware available on the network device e.g. memory and flash storage available.

For example, Cisco IOS XE running on a Catalyst 9000 switch reserves dedicated memory and CPU resources for application hosting. By reserving memory and CPU resources, the switch provides a separate execution space for user applications. It protects the switch's IOS XE run-time processes ensuring both its integrity and performance.

Applications must reside in one of the external Solid State Drive (SSD) storage options (USB or M2 Serial Advanced Technology Attachment(SATA)) provided by Catalyst

9000 switches. Applications have no access to the internal device flash storage for security and flash integrity.

> The external SSD storage is required and shared (not reserved) between Cisco IOS XE and hosted applications.

TABLE *Catalyst 9000 Application Hosting Hardware Resources*

Platform	Memory (GB)	CPU (cores)	USB Storage (GB)		M2 SATA Storage (GB)
			USB 2.0 Front	USB 3.0 Back	
Catalyst 9300	2	1 x 1.8GHz	16	120	N/A
Catalyst 9400	8	1 x 2.4GHz	16	N/A	960
Catalyst 9500	8	1 x 2.4GHz	16	120	N/A
Catalyst 9500 high-performance	8	1 x 2.4GHz	N/A	N/A	960

Resource allocation and utilization can be monitored using any of the management interfaces provided.

The following command shows the total available and used resources for a Catalyst 9300 platform:

```
cat9k#show app-hosting resource
  Disk space:
    Total: 10000 MB
    Available: 10000 MB
  Memory:
    Total: 2048 MB
    Available: 2048 MB
  CPU:
    Total: 29600 units
    Available: 29600 units
```

Networking

A container can have up to four virtual interfaces that are displayed in the container as "eth0", "eth1", "eth2", and "eth3".

Each interface can be attached to either the management interface or a data port. When using a data port, a Virtual Port Group (VPG) needs to be configured. The VPG will serve as a gateway for all the containers and allow them to connect to the IOS XE routing domain through the front panel data ports.

Application Hosting Management

Cisco IOS XE network devices provide several options to easily manage applications:

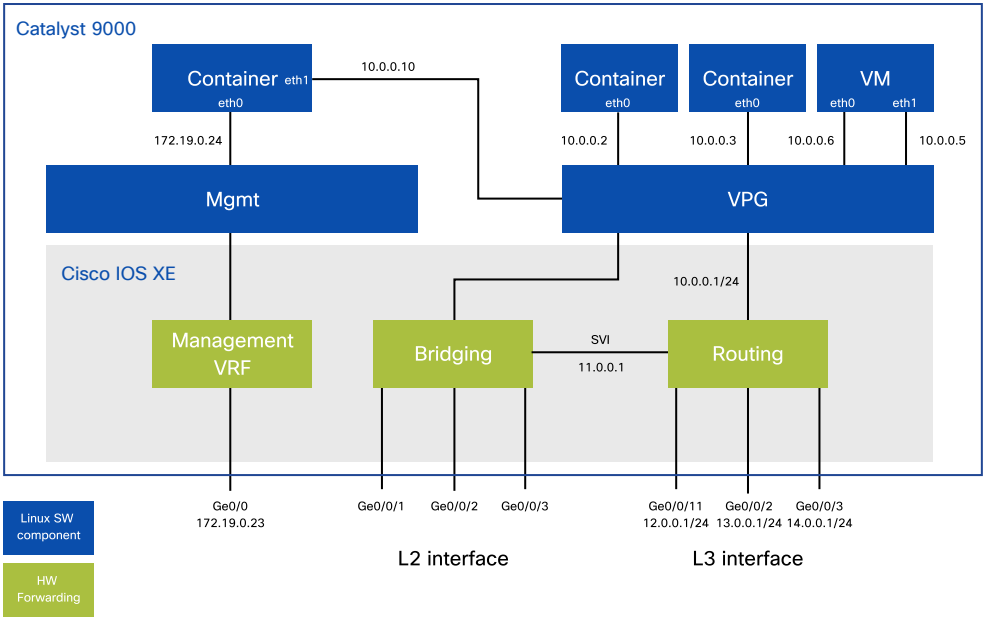
- **Command-line Interface**: Using the new app-hosting CLIs
- **YANG Data Models**: Using NETCONF or RESTCONF protocol
- **WebUI**: The Graphical User Interface provided by Cisco IOS XE to configure and monitor a single device
- **IOx Client**: Python-based client to build and manage IOx applications
- **Cisco DNA Center**: This is the Cisco-recommended solution

Additional benefits of using DNA Center are:

- **Consistent User Interface:** Applications are integrated with the DNA Center UI while re-using their own APIs
- **Integration with DNA Center Assurance:** Assurance can launch hosted applications for monitoring and remediation

Details of a given application can be found using any of the above-mentioned management options.

DIAGRAM *Catalyst 9000 Container Networking*



The following CLI output for a virtual machine includes the current resource allocation and the operational status:

```
cat9k#show app-hosting detail appid wireshark
  State            : RUNNING
  Author           : Cisco Systems, Inc.
  Application
    Type           : vm
    App id         : wireshark
    Version        : 3.4
    Description    : Lubuntu based Wireshark
  Resource reservation
    Memory               : 1792 MB
    Disk                 : 10 MB
    CPU                  : 6600 units
    VCPU                 : 2
  Network interfaces
  eth0:
    MAC address:     52:54:dd:cd:0f:d4
    IPv4 address:    172.26.249.202
```

Containers and Virtual Machines

Though Cisco will periodically publish certain services for application hosting, Cisco encourages and supports the deployment of any application that fits within CAF.

In order to enable network operators, partners, and third-party software vendors to build containers for platforms that support CAF, Cisco provides a command-line tool, called "ioxclient" available at Cisco CCO and DevNet free of charge.

ioxclient can be used in three main workflows to build an IOx container:

LXC Container

The required files for an IOx package are:

- **Disk image**: For the LXC container, the disk image can be created using standard Docker tools and a Dockerfile to describe the base image (i.e, Ubuntu, CentOS, Alpine, etc.) and optional applications to be installed.
- **Application descriptor:** YAML file that provides information for building the package such as the application description (author, versioning etc.), container type (LXC), hardware resources, network interfaces, the disk image and startup application.

All the provided files and information are used by the ioxclient to build the LXC IOx container that can be then downloaded and installed in the IOS XE device.

KVM Virtual Machine

For KVM-based Virtual Machines, the workflow is very similar to the previous one but the disk image to be provided has to be in the qcow2 format instead. The application descriptor for a VM is very similar to the LXC container.

Using an existing Virtual Machine running in a virtual environment such as VirtualBox, the related .vmdk file can be converted in qcow2 format using tools like "qemu-img".

Docker Container

While Docker containers are not natively supported in Cisco IOS XE platforms, Cisco provides a reference VM based on Alpine Linux including "dockerd", which allows network engineers to use Docker tools such as Docker Swarm or Kubernetes without having to rebuild containers.

This approach provides network engineers the flexibility to add any kernel module required without relying on modules provided by IOS XE only.

Step-by-step instructions on how to build IOx applications are available on the Cisco DevNet website: *https://developer.cisco.com/site/iox/*

Use Case

A common task for network administrators and operators is troubleshooting and verifying network performance. While the network may not always be the source of performance issues, network operators provide help in identifying the source of the issue.

Performance Monitoring

perfSONAR, the first application available for Catalyst 9000 switches, is an Open Source network measurement toolkit, designed to provide federated coverage of paths. perfSONAR simulates application traffic and provides an application-centric network performance view which:

- differentiates between an application-level issue versus a more generic network-level issue;
- locates the source of the bottleneck; and
- centralizes performance monitoring into a single view, to quickly pinpoint performance issues.

Benefits from running utilities on the network infrastructure include:

- no dedicated parallel infrastructure
- selection of traffic of interest intuitively — no manual configuration of SPAN sessions
- trigger application access to network device traffic automatically



Application Hosting provides the infrastructure to run any type of application on the supported Cisco devices. Cisco does not recommend App Hosting for use cases such as cryptocurrency mining, gaming, and video streaming.

Next Steps

For more information about Application Hosting on Cisco IOS XE devices, refer to Cisco DevNet. DevNet provides documentation on CAF, IOS XE-specific documentation, and Learning Labs with step-by-step instructions for building IOx applications.

Additional Application Hosting references:

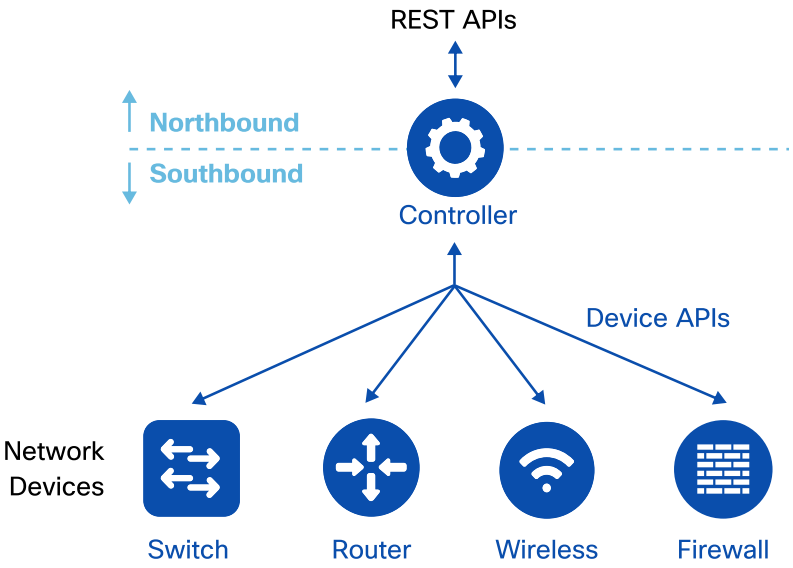
https://developer.cisco.com/site/iox/
https://developer.cisco.com/site/ios-xe/
https://developer.cisco.com/site/networking/

Controllers

Introduction

A network controller is a centralized platform for managing the configuration and operational data of network devices. Controllers often take over management of the network devices, meaning that all interactions with the devices take place through the controllers only. The general idea of controllers is to abstract and centralize administration of individual network devices, reducing or eliminating the need for device-by-device configuration and management.

DIAGRAM *Network Controller Architecture*



Controller APIs

Controllers expose northbound APIs to integrate with other enterprise IT systems, such as Business Intelligence, monitoring, logging and security systems. Northbound APIs are exposed by the REST framework.

Controllers integrate southbound with network elements via NETCONF, but other options exist including RESTCONF, OpenFlow, and the gRPC Network Management Interface (gNMI).

Common Controllers

The following is an overview of controllers commonly used in enterprise networks.

Cisco Digital Network Architecture (DNA) Center

The Cisco DNA Center dashboard provides an intuitive and simple overview of network health and clear drill-down menus for quickly identifying and remediating issues. The network design interface allows quick creation of physical maps and logical views of all network deployment locations. Automation and orchestration provide Zero-Touch Provisioning based on profiles which facilitate remote branch network deployments. Advanced assurance and analytics capabilities use deep insights from devices, streaming telemetry, and rich context to deliver an uncompromised experience while proactively monitoring, troubleshooting, and optimizing wired and wireless networks.

Key features of DNA Center are:

- **simple design:** network design using intuitive workflows, starting with locations where network devices will be deployed
- **secure policy:** define user and device profiles that facilitate highly secure access and network segmentation based on business needs
- **automated provisioning:** policy-based automation to deliver services to the network based on business priority and to simplify device deployment
- **proactive assurance:** deep insights with rich context to deliver a consistent experience and proactive optimization of the network
- **programmability and APIs:** support for Representational State Transfer (REST) APIs at the northbound layer for programmability

More information about Cisco DNA Center is available from *https://www.cisco.com/go/dnacenter*

Cisco Network Service Orchestrator (NSO)

Cisco Network Services Orchestrator is an orchestration platform for hybrid networks.

Northbound, NSO provides a number of interfaces including NETCONF and REST based on services defined using YANG data models to automate network services.

NSO supports various southbound network management protocols, including NETCONF and CLI, to maintain multi-vendor device configurations in the network.

Some of the NSO advantages are:

- **device abstraction:** abstracts device details and focuses on the network services that are being managed
- **network-wide transactions:** enables network-wide transactional service provisioning
- **simplified automation:** provides a simpler automation workflow
- **multi-vendor:** makes possible management of multi-vendor environments

OpenDaylight (ODL)

ODL is an open source controller which assists control plane applications to interact with network devices through a variety of southbound protocols including NETCONF and OpenFlow. ODL architecture is based on a YANG data model-driven service abstraction layer and supports the northbound interfaces of NETCONF, REST and RESTCONF.

Cisco and others are collaborating on this open source project. ODL is available for download or via a commercial distribution.

Why Use a Controller?

Controller-level and device-level automation are both valid choices for managing a network. Device-level automation provides more control but requires more effort to manage. Conversely, controllers do not allow the granular level of control of device-level automation and are easier to manage.

For example, it is possible to deploy the individual features of a Software-Defined Access fabric device-by-device by using either NETCONF or even CLI, with configuration management tools such as Ansible and Puppet. This allows the user more granular control than providing the configuration through DNA Center. However, configuring a large number of devices for a fabric in this way would require more work to define the configuration parameters, data models, and to ensure that proper validation of configurations takes place. DNA Center would do these automatically.

It is also possible to use a combination of controller-based and device-based automation. For example, DNA Center could be used to provision and manage most of the settings in the network, but certain scripts could be used to collect certain pieces of operational data.

While the controller still manages the devices and provides a layer of abstraction to the network operator, it is still possible to use scripts or other tools to automate the network using the northbound REST APIs available to the controller.

DevOps and NetDevOps

Introduction

Traditionally, software development has followed a waterfall process where all requirements and features are defined at the beginning. Major releases deliver significant software changes and new capabilities. However, the time between major releases can be months or years.

In recent years, there has been a major shift toward an Agile Software Development methodology, based on small increments that minimize the amount of up-front planning and design. Short, iterative bursts of development, or sprints, are key characteristics of this methodology. Each iteration involves a team working across plan ning, analysis, design, coding, unit testing, and acceptance testing.

The Agile Software Development process can create friction between development and operations teams due to different priorities as shown in the table below:

TABLE *Development vs Operations Priorities*

Development	Operations
Frequent Changes	Stability
Feature Development	Business Operations
Creativity	Productivity
" Let me try this!"	" Please no change!"

Development and Operations (DevOps) is a set of practices which reduces the time between developing a change to a system and rolling it out in production while ensuring high quality. This requires the adoption of a new culture, new team structure, and new processes and workflows which aim to eliminate the friction between Development and Operations.

NetDevOps is the integration of Networking teams with DevOps teams. This applies the values of automation, continuous integration and scalability to the network infrastructure.

NetDevOps requires the network engineer to master networking skills, learn scripting and the tools necessary for Continuous Integration/Continuous Delivery (CI/CD).

Continuous Integration and Delivery

Continuous Integration and Continuous Delivery (CI/CD) reduces the time to deploy changes in production and increases the confidence that changes will be successful. It is enabled through an automated testing process and integration with source control systems for an easy rollback in case failures occur. The benefits of implementing CI/CD are shorter development cycles, a faster pace of innovation, and a lower total cost.

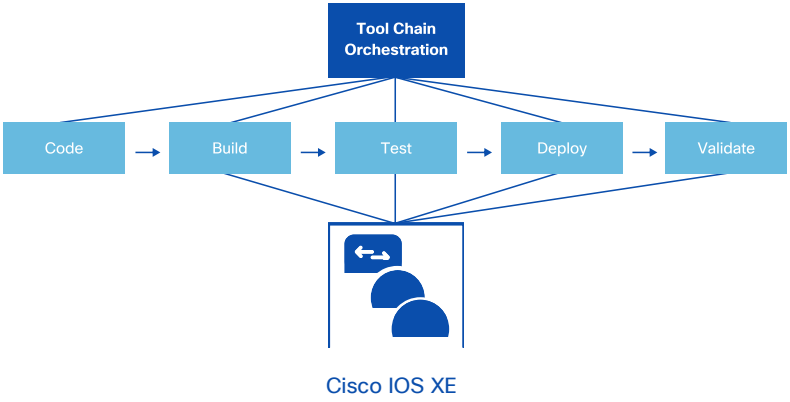
Continuous Integration merges all developer working copies to a shared code repository several times a day. It includes automated testing to ensure changes do not break the "application" or in the case of Cisco IOS XE, the network.

CD stands for either Continuous Delivery or Continuous Deployment. With Continuous Delivery, the changes are manually released according to schedule. With Continuous Deployment, the entire deployment process is fully automated.

Most network operators are expected to implement a Continuous Delivery model and move to Continuous Deployment once they are confident with the entire process.

The steps of the CI/CD workflow are illustrated in the diagram below.

DIAGRAM *CI/CD Workflow*



Jenkins, GitLab, Trevis CI, Drone, and Bamboo are examples of the Tool Chain Orchestrator which manages the entire workflow:

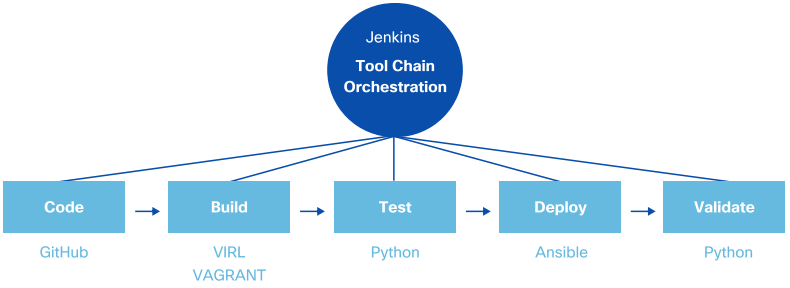
- **Code**: Engineers create a branch on the shared source control repository, publish the proposed changes, and merge the updates into the master branch. Source control tools such as Git, GitHub, GitLab and Gogs are the main tools used in this phase.
  GitHub is the most popular tool and merges can be requested by issuing a Pull Request (PR) for the master branch that can automatically trigger the next phase.
- **Build**: Upon merging of new code, a test environment is built from scratch or reconfigured, replicating the production environment. For example, GitHub services can be used to send a webhook into Jenkins to create a Cisco VIRL instance.
- **Test:** The changes are tested in the environment created in the Build phase. Depending on tests results, the process moves to the next phase or notifies the engineer about the failure and stops the workflow. For example, Jenkins can trigger automated tests for data validation, reachability and performance by leveraging common tools such as Python scripts and the iPerf tool.
- **Deploy:** If the tests are successful, the changes are deployed in production. Typically the deployment is performed using a configuration management tool such as Ansible or Puppet, and can be either manual or automatic, depending on whether a Continuous Delivery or a Continuous Deployment methodology is implemented.
- **Validate**: After the changes are deployed, tests are performed to ensure everything is working in production. If the tests fail, the configurations are automatically rolled back. Test results are stored for tracking purposes and notifications are sent to the DevOps team using collaboration tools such as Cisco Webex Teams, and ticketing systems such as ServiceNow, Jira or email.

DevOps Tools

DevOps engineers need to identify proper tools. There are many Open Source and commercial tools available. A comprehensive list of these tools can be found in the Periodic Table of DevOps Tools built by XebiaLabs:*https://xebialabs.com/periodic-table-of-devops-tools/*

The image below shows the most common tools used in each phase of the CI/CD workflow.

DIAGRAM *CI/CD Workflow Tool Examples*



Commonly Used DevOps Tools

Jenkins

Jenkins is a self-contained, open source Tool Chain Orchestrator to automate building, testing, and delivering or deploying software.

GitHub

GitHub is a web-based hosting service for software version control using Git. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

GitHub offers plans for both private repositories and free public accounts to host open source projects.

Virtual Internet Routing Lab (VIRL)

VIRL is a Cisco software-based network simulation framework that makes possible the rapid design, configuration, and testing of network infrastructure. It assists virtual network devices running Cisco operating systems, to integrate with physical network devices, third-party virtual machines and servers.

VIRL supports building an accurate development and test environment which closely matches enterprise network infrastructures. More information is available at *http://virl.cisco.com*

Vagrant

Vagrant is a tool for building and managing virtual machine environments. With Vagrant, developers can build a virtual environment using a simple configuration file. The Vagrant file is used to create virtual machines from basic templates (known as "boxes"), customize the operating system, install software, configure networking and more. The network engineer can also quickly create test virtual environments for Cisco IOS XE, IOS XR, and NX-OS.

More information can be found at *https://www.vagrantup.com*

Configuration Management Tools (CMT)

CMT automate systems and applications consistently at scale. The benefits of such tools are:

- a consistent approach across different vendors and operating systems
- easy integration with software version control systems

- a simple way to collect hardware and software device information
- an intent-based configuration approach
- no changes are made if the system, application, or device is already in the desired state ("idempotency")

In networking, CMTs were initially used only to automate data center networks, but are now being used in enterprise networks as well.

CMT fall into two categories: agent-based or agent-less. Agent-based architectures require a software agent to be installed on the managed device. Agent-less architectures are more popular in enterprise networks where third party software is typically not installed on network devices.

The two most popular CMTs are:

- **Ansible:** an agent-less open source CMT which allows the operator to describe the automation jobs in an easily readable format. Ansible configurations are defined in files called "playbooks" which are written using YAML. Ansible supports Cisco IOS XE devices using CLI and NETCONF-based configurations.
- **Puppet:** currently requires an agent on the managed device but is moving towards an agent-less architecture for network devices. Puppet will support Cisco IOS XE devices and will be based on CLI and NETCONF.

Webex Teams

Webex Teams is a collaboration environment for com- munications between team members. It also has integrations with bots and Webhooks. Conversations in Webex Teams occur in virtual meeting rooms. Webex Team REST APIs provide easy integra tions with CI/CD workflows and support notifications for ChatOps rooms.

Next Steps

The following Cisco DevNet GitHub repository provides a reference for building CI/CD workflows: *https://github.com/CiscoDevNet/iosxe-ci-cd*

Cisco provides abstractions for Cisco Validated Designs at *https://cs.co/validated_design*

Ansible Playbooks for L2 Cisco Validated Design (CVD) configurations are available on GitHub at *https://github.com/CiscoDevNet/cvd-config-templates*

Appendices

Additional Resources

The following list includes some resources to continue learning and using programmability within the network.

Cisco Live!

Attending Cisco Live! provides great opportunities to increase personal knowledge of Cisco products. More information about this event can be found at *https://www.ciscolive.com*

Recordings of previous Cisco Live! sessions can be accessed online from the Cisco Live On-Demand Library at *https://www.ciscolive.com/global/on-demand-library*

The following programmability sessions are recommended:

- DEVNET-1693: Model-Driven Telemetry for Cisco IOS XE
- DEVNET-1801: Insights into your WLC with Wireless Streaming Telemetry
- DEVNET-1828: Cisco's Open Device Programmability Strategy
- DEVNET-2203: Build a Network Configuration CICD Pipeline
- DEVNET-2556: Dive into Leveraging Python on Cisco IOS XE
- BRKCRS-1450: Introduction to Catalyst Programmability
- BRKCRS-2004: Application Hosting and Model-Driven Telemetry on Cisco IOS XE
- BRKCRS-2451: Scripting Catalyst Switches
- BRKSDN-2935: From Zero to Network Programmability in 90 minutes

Cisco DevNet

Cisco DevNet is a program that helps developers and network engineers write applications and develop integrations with Cisco products, platforms and APIs. Some recommended resources include:

- Network Programmability Basics – expert-led video series introducing network programmability *https://developer.cisco.com/video/net-prog-basics*
- Network Programmability Learning Labs – guided learning platform for network programmability *https://learninglabs.cisco.com/tracks/iosxe-programmability*
- IOS XE Sandbox – "Always On" Sandbox provides an environment for developers and network engineers to build and test their applications and scripts *https://devnetsandbox.cisco.com/*
- Developer Support Spark Chat *https://developer.cisco.com/spark-chat/*)
- Knowledge Base and Community *https://devnetsupport.cisco.com*
- Events – in-person events where developers and network engineers can listen, learn and practice with hands-on training. More information about events can be found at *https://devnetevents.cisco.com*

dCloud

Cisco dCloud is a free online platform for evaluating Cisco solutions and products. It provides an extensive catalog of labs, demos, training and Sandboxes for a variety of Cisco technologies including IOS XE programmability labs. Some labs have open access, while others may require a reservation.

dCloud can be extended to an existing enterprise network with the use of a VPN router. The lab environment can also be shared with a team. More information can be found at *https://dcloud.cisco.com*

DIAGRAM *dCloud Solution Catalog*



Build Your Own Programming Environment (BYOE)

One of the most important aspects of acquiring new skills is practice. A critical tool is a developer environment in which to learn, test and practice programmability skills. DevNet has instructions for building a Python environment on a Windows/Mac/Linux laptop.

The lab is called LM-4002: Event Preparation for DevNet Express for DNA v2.1 available at *https://learninglabs.cisco.com/tracks/devnet-express-dna*

Cisco Training and Certifications

Training

Programming for Network Engineers (PRNE) – Learn how to manage a network more efficiently with Network Programmability and develop Python programming fundamental skills. PRNE can be accessed at *https://cs.co/training_prne*

Certifications

Cisco Network Programmability Design and Implementation Specialist (300-550 NPDESI) - Addresses the evolving role of network engineers towards more programmability, automation, and orchestration, enabling them to leverage the powerful level of abstraction provided by controller-based architectures and device APIs to create real added value. More information about this certification can be found at *https://cs.co/training_npdesi*

Cisco Network Programmability Developer Specialist (300-560 NPDEV) – Addresses software developers looking to automate network infrastructure and/or utilize APIs and toolkits to interface with SDN controllers and individual devices. More information about this certification can be found at *https://cs.co/training_npdev*

Acronyms

AAA – Authentication, Authorization, and Accounting

ACL – Access Control List

API – Application Programming Interface

BER – Basic Encoding Rules

BGP – Border Gateway Protocol

BI – Business Intelligence

CAF – Cisco Application-Hosting Framework

CCO – Cisco Connection Online

CD – Continuous Delivery or Continuous Deployment

CDP – Cisco Discovery Protocol

CI – Continous Integration

CLI – Command-line Interface

CMT – Configuration Management Tools

CoS – Class of Service

CPU – Central Processing Unit

CRUD – Create, Read, Update and Delete

CVD – Cisco Validated Design

CVE – Common Vulnerabilities and Exposures

CVS – Concurrent Versions System

DB – Database

DevOps – Development and Operations

DHCP – Dynamic Host Configuration Protocol

DNA – Digital Network Architecture

DNA-C – Digital Network Architecture Center

DNS – Domain Name System

DoS – Denial of Service

EEM – Embedded Event Manager

EIGRP – Enhanced Interior Gateway Routing Protocol

ELK – Elasticsearch Logstash and Kibana

ETA – Encrypted Traffic Analytics

FTP – File Transfer Protocol

GPB – Google Protocol Buffer

gNMI – gRPC Network Management Interface

gRPC – gRPC Remote Procedure Call

GUI – Graphical User Interface

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IEEE – Institute of Electrical and Electronics Engineers

IETF – Internet Engineering Task Force

IoT – Internet of Things

IOx – IOS + Linux

IP – Internet Protocol

IT – Information Technology

JRE – Java Runtime Environment

JSON – JavaScript Object Notation

KPI – Key Performance Indicator

KVM – Kernel-based Virtual Machine

LXC – Linux Container

MAC – Media Access Control

MIB – Management Information Base

NACM – NETCONF Access Control Model

NED – Network Element Driver

NETCONF – Network Configuration Protocol

NetDevOps – Network Development and Operations

NMS – Network Management System

NOC – Network Operation Center

NPDESI – Cisco Network Programmability Design and Implementation Specialist

NPDEV – Cisco Network Programmability Developer Specialist

NSO – Network Service Orchestrator

NVRAM – Non-Volatile Random-Access Memory

OC – OpenConfig

ODL – OpenDaylight

OID – Object Identifier

OS – Operating System

OSPF – Open Shortest Path First

PIP – Pip Installs Python

PnP – Plug and Play

PR – Pull Request

PRNE – Programming for Network Engineers

PXE – Pre-boot Execution Environment

QoS – Quality of Service

RADIUS – Remote Authentication Dial-In User Service

REST – Representational State Transfer

RESTCONF – REST Configuration

RFC – Request For Comment

RAM – Random Access Memory

RMA – Return Material Authorization

ROMmon – Read-Only Memory Monitor

RPM – Red Hat Package Manager

SATA – Serial Advanced Technology Attachment

SN – Serial Number

SNMP – Simple Network Management Protocol

SPAN – Switch Port Analyzer

SSD – Solid State Drive

SSH – Secure Shell

TFTP – Trivial File Transfer Protocol

URI – Uniform Resource Identifier

USB – Universal Serial Bus

VIRL – Virtual Internet Routing Lab

VM – Virtual Machine

VPG – Virtual Port Group

VRF – Virtual Routing and Forwarding

VTY – Virtual Terminal Line

WAN – Wide Area Network

WLC – Wireless LAN Controller

XML – eXtensible Markup Language

YANG – Yet Another Next Generation

YAML – YAML Ain't Markup Language

YUM – Yellowdog Updater Modified

YDK – YANG Development Kit

ZTP – Zero-Touch Provisioning