Programming and Automating Cisco Networks
A Guide to Network Programmability and Automation in the Data Center, Campus, and WAN

Ryan Tischer, CCIE No. 11459
Jason Gooley, CCIE No. 38759

ciscopress.com
Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Feedback Information

At Cisco Press, our goal is to create in-depth technical books of the highest quality and value. Each book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from the professional technical community.

Readers’ feedback is a natural continuation of this process. If you have any comments regarding how we could improve the quality of this book, or otherwise alter it to better suit your needs, you can contact us through email at feedback@ciscopress.com. Please make sure to include the book title and ISBN in your message.

We greatly appreciate your assistance.

Editor-in-Chief: Mark Taub
Product Line Manager: Brett Bartow
Alliances Manager, Cisco Press: Ronald Fligge
Managing Editor: Sandra Schroeder
Development Editor: Ellie C. Bru
Project Editor: Mandie Frank
Copy Editor: Lori Martinsek

Technical Editor(s): Joe Clarke, Omar Sultan
Editorial Assistant: Vanessa Evans
Cover Designer: Chuti Prasertsith
Composition: codeMantra
Indexer: Erika Millen
Proofreader: Kamalakannan
About the Authors

Ryan Tisher, CCIE No. 11459 is a Technical Solution Architect at Cisco where he focuses on SDN, Cloud, and network programmability. He has worked in IT for 20 years, specifically focused on design, deployment, and operations of networking technologies. Ryan holds a BA in Information Technology from the University of Massachusetts, Lowell and a MS in Network Engineering from Depaul University. Ryan lives with his wife and children in the Milwaukee, WI area. Ryan blogs at http://Policyetc.com.

Jason Gooley, CCIE No. 38759 (R&S & SP), is a very enthusiastic engineer that is passionate about helping others in the industry succeed. Jason has more than 20 years of experience in the Information Technology and Telecommunications industry. Jason currently works at Cisco as a Strategic Systems Engineer where he specializes in SD-WAN, campus, and data center network design. In addition, Jason works with Learning@Cisco on certification development, mentoring, and training. Jason is also a Program Committee member and organizer for the Chicago Network Operators Group (CHINOG). Jason lives in Illinois with his wife Jamie and their daughter Kaleigh.
About the Technical Reviewers

Joe Clarke, CCIE No. 5384 is a Global TAC engineer. He has contributed to network management products and technologies by finding and fixing bugs, as well as implementing maintenance and troubleshooting components in Cisco Prime.

Joe helps to support and enhance the embedded automation and programmability technologies, such as the Embedded Event Manager, Tcl, NETCONF/RESTCONF, and ONE Platform Kit (onePK). Joe is a top-rated speaker at Cisco's annual user conference, CiscoLive!, as well as certified as a Cisco Certified Internetworking Expert, Certified Java Programmer, and VMware Certified Professional. Joe provides network consulting and design support for the Internet Engineering Task Force (IETF) conference network infrastructure deployments. He has authored numerous technical documents on Cisco network management, automation, and programmability products and technologies. Joe is co-author of more than 20 Cisco patents. He is an alumnus of the University of Miami and holds a Bachelor of Science degree in computer science. Outside of Cisco, Joe is a member of the FreeBSD project. He is a committer in the project focusing mainly on the GNOME Desktop. He also maintains the FreeBSD ports Tinderbox application, which facilitates the automated packaging and testing of FreeBSD third-party ports.

Omar Sultan currently leads a team of sales managers and product managers focused on Cisco’s web and cloud customers. At Cisco since 1999, Omar has focused on helping the company successfully enter new markets and is currently focused on the software and hardware technologies that underpin web-scale infrastructure. A geek at heart, Omar has been involved with IT since VAXes roamed the earth. Omar has been involved in every aspect of IT from cabling to coding to systems and networking, which has left him the perspective that data centers should really be viewed as their own class of quirky complex life forms.
Dedications

Ryan Tischer:

This book is dedicated to my wife Jennifer and my children Madeline, Alexander, and Elaina. When the road gets rough, you are the reason I do not give up. When scary arrives, you are the source for my courage. When good enough is reached, you make me push for better. When things don’t go my way, you make me substitute my cuss words.

ILMF4EVER

Special thank you to my parents—Stop saying I turned out all right; there’s still time.
To my friends—I know the best place for chili.
A message to my children—
Whatever your life has in store for you, be …
curious
passionate
thoughtful
Break ground and glass
Be anything but boring.
—Ryan

Jason Gooley:

I would like to dedicate this book to my family. To my wife Jamie for being so endlessly supportive of me with my various “projects.” Without you, I would not have been able to make it this far. To my daughter Kaleigh, who at the time of this writing is just 15 months old: It is extremely difficult to leave your side when all I want to do is spend time with you. I feel like I have already missed so much just writing this paragraph! To my father and brother for always having my back and believing in me. To my late mother, you have been the guiding light that has kept me on the right path.
Acknowledgments

Ryan Tischer:

I'd like to give special recognition to the amazing engineers, managers, sales teams, and customers I have the privilege of working with. I am humbled to be a part of this community, and I fully recognize that without your inspiration, encouragement, and knowledge, this book would not be possible. I have been truly blessed to have managers and co-workers who believe in me, told me when I screwed up, and gave me the opportunity of a lifetime. Special thank you to INSBU for building wickedly-cool products and letting me play.

A big thank you to Joe Clark, Omar Sultan, Brett Bartow, and Eleanor Bru for their amazing work on this book.

Finally, I'd like to thank my co-author Jason Gooley. I approached him with this project at the very last minute, and he's worked very hard to keep the book on time, while not sacrificing technical depth or content.

Jason Gooley:

First, thank you to Brett Bartow, Eleanor Bru, and the rest of the Cisco Press team for all of the support during creation of this book. It was a pleasure to work with such an amazing group of professionals.

I would like to thank the entire Cisco Commercial Midwest Select Operation for supporting me during this process. Thank you to my manager and all of my teammates on the Illinois Select team for the continued reinforcement of this project.

A special thanks to Anthony Sequeira, Keith Barker, Andre Laurent, and Luke Kaelin for all the mentoring and words of encouragement. I can't thank you enough for all of your support over the years.

Thank you, Ryan, for giving me the opportunity to write this book with you. It has been an absolute blast, and I am honored to be a part of it.

Finally, I would like to thank all my friends and family who have patiently waited for me to finish this project, so I would be able to go outside and play. You know who you are!
Contents at a Glance

Introduction xviii

Section I Getting Started with Network Programmability
Chapter 1 Introduction: Why Network Programmability 1
Chapter 2 Foundational Skills 13

Section II Cisco Programmable Data Center
Chapter 3 Next-Generation Cisco Data Center Networking 67
Chapter 4 On-Box Programmability and Automation with Cisco Nexus NX-OS 83
Chapter 5 Off-Box Programmability and Automation with Cisco Nexus NX-OS 125
Chapter 6 Network Programmability with Cisco ACI 159

Section III Cisco Programmable Campus and WAN
Chapter 7 On-Box Automation and Operations Tools 215
Chapter 8 Network Automation Tools for Campus Environments 255
Chapter 9 Piecing It All Together 303

Index 307
Contents

Introduction xviii

Section I Getting Started with Network Programmability

Chapter 1 Introduction: Why Network Programmability 1
What Is Network Programmability 3
  Network Programmability Benefits 4
  Simplified Networking 4
  Network Innovation with Programmability 4
  Cloud, SDN, and Network Programmability 6
  SDN 8
  Is Programmability a New Idea? 9
  Network Automation 10
  Automation Example 11
Summary 11

Chapter 2 Foundational Skills 13
Introduction to Software Development 13
  Common Constructs—Variables, Flow Control, Functions, and Objects 15
  Variables 15
  Flow Control—Conditions 17
  Flow Control—Loops 18
  Functions 18
  Objects 19
  A Basic Introduction to Python 20
  More on Strings 22
  Help! 23
  Flow Control 24
  Python Conditions 24
  Python Loops 25
  While Loop 26
  Python Functions 28
  Python Files 29
  Importing Libraries 30
  Installing Python Libraries 30
  Using PIP 31
Chapter 4  On-Box Programmability and Automation
with Cisco Nexus NX-OS  83

Open NX-OS Automation—Bootstrap and Provisioning  83
  Cisco POAP  83
  Cisco Ignite  87
  Using Ignite  87
  NX-OS iPXE  88

Bash  88
  Bash Scripting  89
  Bash Variables, Conditions, and Loops  89
  Bash Arithmetic  90
  Bash Conditions and Flow Control  91
  Bash Redirection and Pipes  94
  Working with Text in Bash  96

Awk  98

Bash on Nexus 9000  99
  ifconfig  101
  Tcpendump  101
  ethtool  103
  Run a Bash Script at Startup  103
  Bash Example—Configure NTP Servers at boot  106

Linux Containers (LXC)  106
  Network Access in Guestshell  109
  Installing Applications in Guestshell  110
  Puppet Agent Installation in Guestshell  111
  NMap Installation in Guestshell  111
  Embedded Nexus Data Broker  111
  Nexus Embedded Event Manager  112
  EEM Variables  113
  On-box Python Scripting  113
  Using the NX-OS Python CLI Library  115
  Using NX-OS Cisco Python Library  116
  Non-Interactive Python  118
  Cisco or CLI Package?  118

On-Box Python—Use Cases and Examples  118
  EEM Neighbor Discovery  121

Summary  124
Reader Services

Register your copy at www.ciscopress.com/title/ISBN for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to www.ciscopress.com/register and log in or create an account.* Enter the product ISBN 9781587144653 and click Submit. Once the process is complete, you will find any available bonus content under Registered Products.

*Be sure to check the box that you would like to hear from us to receive exclusive discounts on future editions of this product.
Icons Used in This Book

Command Syntax Conventions

The conventions used to present command syntax in this book are the same conventions used in the IOS Command Reference. The Command Reference describes these conventions as follows:

- **Boldface** indicates commands and keywords that are entered literally as shown. In actual configuration examples and output (not general command syntax), boldface indicates commands that are manually input by the user (such as a show command).

- **Italic** indicates arguments for which you supply actual values.

- Vertical bars (|) separate alternative, mutually exclusive elements.

- Square brackets ([ ]) indicate an optional element.

- Braces ({ }) indicate a required choice.

- Braces within brackets ([[ ]]) indicate a required choice within an optional element.
Introduction

This book was designed with the focus on utilizing Cisco ACI Cisco Nexus 9000, Cisco UCS Director, Cisco (JSON), Python, Linux, Cisco APIC-EM, ConfD, and Data Models in a production environment as effectively as possible. Industry leaders were consulted for technical accuracy throughout this book.

Who Should Read This Book?

This book is designed for those network engineers and operators who want to implement, manage, and maintain Cisco networking solutions in modern environments. This book discusses automation and programming tools and techniques across the Cisco data center, campus, and LAN and WAN technologies.

How This Book Is Organized

Chapter 1, “Introduction: Why Network Programmability:” Network programmability can solve business problems, reduce operating expenses and increase business agility. Current network management is slow and prone to errors because it’s a closed, box-by-box, CLI-driven system that requires constant and expensive attention. Network programmability serves as a tool kit to automate network configurations and troubleshooting, significantly reducing nonoperational states. Additionally network programmability allows the network to participate or add value to dynamic application environments, that is, DevOps, web, security, by facilitating a tight bond between applications and infrastructure.

Chapter 2, “Foundational Skills:” A basic introduction into software engineering and DEVOPS.

Chapter 3, “Next-Generation Cisco Data Center Networking:” This chapter discusses Cisco portfolio and where the reader could possibly implement network programmability and automation.

Chapter 4, “On-Box Programmability and Automation with Cisco Nexus NX-OS:” This chapter discusses writing software designed to run on the Nexus switch.

Chapter 5, “Off-Box Programmability and Automation with Cisco Nexus NX-OS:” This chapter discusses writing software to run on other systems and access Nexus switches remotely.

Chapter 6, “Network Programmability with Cisco ACI:” Chapter 6 discusses writing software to interact and enhance Cisco ACI.
Chapter 7, “On-Box Automation and Operations Tools:” This chapter discusses some of the automation and operations tools that are available on many Cisco platforms.

Chapter 8, “Network Automation Tools for Campus Environments:” Automation tools can be off-box as well as on-box. This chapter covers some of the tools available for Cisco campus networks including SDN, controllers, and more.

Chapter 9, “Piecing It All Together:” This chapter summarizes the contents of this book by giving our perspective on the many tools that are available to interact with Cisco networks.
This page intentionally left blank
Automation for daily tasks is something that most network engineers rely on to handle their daily workload. However, there are many network engineers under the impression that new software or management tools with a steep learning curve must be purchased in order to accomplish such automation. This leads to automation tools often times getting overlooked or put aside. Other common drivers for not using automation tools are due to budget restraints, varying skill sets, and unfamiliarity with the different tools that are available. The good news is that there are many automation tools natively available on most Cisco IOS platforms. For instance, on most Cisco Catalyst switches, there are tools built into the operating system’s command line interface (CLI) that allow the programmability of these devices automatically. This allows for the automation of large number of common tasks. For example, a network engineer could build a set of custom templates or macros that would apply various configuration parameters to particular ports on a switch, based on the types of devices that are connected to those specific ports. This chapter will cover the following on-box automation tools in greater detail:

- Auto SmartPorts
- AutoConf
- Auto Security
- AutoQoS
- Smart Call Home
- Tcl Shell
- Embedded Event Manager (EEM)

**Note** For brevity, all configuration examples and outputs in this chapter are displayed in IOS only. IOS XE and IOS XR outputs are not included in this chapter.
Automated Port Profiling

These types of automation tools are especially important when it comes to scale. Imagine a network team that handles an entire enterprise campus LAN. Commonly, there are only a select few network engineers who have access to the network switches and are authorized to make any configuration changes. These engineers are usually very busy and have a finite amount of time to work on daily moves, additions, and changes (MACs). From a business perspective, this greatly hinders the capability of being able to fluidly and dynamically move users around an office environment. For example, a user moves from one department to another and takes their IP phone with them. This would result in a network engineer having to get involved and reprogram the switch port that the user is going to be connecting to. Often, these users are moved by a help desk team without notifying the network engineering team of the move. If the new port wasn't properly provisioned prior to the user moving, then the user may not be able to connect to the network and perform their job.

There are many settings that need to be applied to a switch port in order for an IP phone to operate properly. Some of the more common switch port settings for an IP phone are:

- Power over Ethernet (PoE) settings
- Voice VLAN configuration
- Quality of Service (QoS) settings
- Data VLAN configuration
- Speed/Duplex settings

The following sections of this chapter will cover some of the simple, yet powerful tools that are included within most of the Cisco Catalyst switches. These are tools, available today, that can automate many configuration tasks, reduce downtime, and increase agility.

AutoSmart Ports

AutoSmart Ports (ASP) are an IOS tool that allows you to consolidate many of the necessary port settings for various device types into an automated process that can be applied to a single port or a series of ports. AutoSmart ports use a macro-based mechanism that commonly uses CDP and LLDP to discover the physical device type that is connected to a switch port. Once the device type is determined, the switch will then check to see if a corresponding macro is defined that matches the specific device type that was connected. If the device type is known and there is a macro definition for it, the switch will then automatically provision the port, based on the settings defined in the macro. This will significantly reduce the amount of time needed to establish connectivity to users who move around the environment or for new users who are being brought on board for the first time. Figure 7-1 outlines the process for what happens when a Cisco IP phone device is connected to a Catalyst switch while AutoSmart Ports are enabled.

Note  AutoSmart Ports are available in IOS 12.2(55)SE or later.
One of the main advantages of AutoSmart ports is that the switches contain predefined macros that can be modified to suit your environment. In addition, you can also customize those predefined macros to include all the necessary parameters for your specific environment. Table 7-1 shows a list of some of the predefined device-specific macros that are available in most Cisco Catalyst switches.

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Macro Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access-point</td>
<td>Auto configuration information for the autonomous access point</td>
</tr>
<tr>
<td>ip-camera</td>
<td>Auto configuration information for the video surveillance camera</td>
</tr>
<tr>
<td>lightweight-ap</td>
<td>Auto configuration information for the lightweight access point</td>
</tr>
<tr>
<td>media-player</td>
<td>Auto configuration information for the digital media player</td>
</tr>
<tr>
<td>Phone</td>
<td>Auto configuration information for the phone device</td>
</tr>
<tr>
<td>Router</td>
<td>Auto configuration information for the router device</td>
</tr>
<tr>
<td>Switch</td>
<td>Auto configuration information for the switch device</td>
</tr>
</tbody>
</table>

**Enabling AutoSmart Ports on a Cisco Catalyst Switch**

In order to enable AutoSmart Ports on a Cisco Catalyst switch, you must follow the steps illustrated in the following example. Another key advantage of this specific automation tool is that it takes a single command to enable to macro functionality.

```
Switch> enable
Switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# macro auto global processing
Switch(config)# end
Switch#
```
Occasionally, predefined macros contain most of the desired settings that are needed without requiring any modification to the macro. In some cases, however, customizing a macro to fit your needs is a better alternative. Customized macros are commonly deployed when more granular configurations are required. For example, a customized macro may be one that not only changes voice and data VLANs, but can also be used to configure quality of service (QoS) settings and other various options. The following example lists the default settings of the Cisco IP phone macro. This can be seen with the `show macro auto device phone` command.

```
Switch# show macro auto device phone
Device:phone
Default Macro:CISCO_PHONE_AUTO_SMARTPORT
Current Macro:CISCO_PHONE_AUTO_SMARTPORT
Configurable Parameters:ACCESS_VLAN VOICE_VLAN
Defaults Parameters:ACCESS_VLAN=1 VOICE_VLAN=2
Current Parameters:ACCESS_VLAN=1 VOICE_VLAN=2
```

**Note** To view the entire list of predefined macros that are available in a Cisco Catalyst switch, issue the `show shell functions` command.

The following output illustrates the configuration steps that are necessary to customize and trigger a predefined macro. In this example, the macro, when applied, will change the voice and data VLANs for a port when Cisco IP phone is connected.

```
Switch# configure terminal
Switch(config)# macro auto execute CISCO_PHONE_EVENT builtin CISCO_PHONE_AUTO_SMARTPORT ACCESS_VLAN=11 VOICE_VLAN=13
Switch(config)# macro auto global processing
Switch(config)# exit
```

To verify this macro is properly modified with the new VLAN assignments, issue the `show shell triggers` command from the EXEC prompt of the CLI. The following snippet shows the output from the `show shell triggers` command.

```
Switch# show shell triggers
User defined triggers
------------------
Built-in triggers
------------------
Trigger Id: CISCO_PHONE_EVENT
Trigger description: Event for ip-phone macro
Trigger environment: ACCESS_VLAN-11 VOICE_VLAN-13
Trigger mapping function: CISCO_PHONE_AUTO_SMARTPORT
```
Other common event triggers that can be viewed and modified are:
- Trigger Id: CISCO_ROUTER_EVENT
- Trigger Id: CISCO_SWITCH_EVENT
- Trigger Id: CISCO_WIRELESS_AP_EVENT
- Trigger Id: CISCO_WIRELESS_LIGHTWEIGHT_AP_EVENT

In certain cases, the device you connect to the switch may not be able to use CDP or LLDP to identify itself to the switch. In these instances, you can create a custom macro that uses a BASH-like language syntax. Another interesting use case utilizes the MAC address OUI to identify and properly configure various devices on the switch. The following example shows a custom macro for a printer, using the MAC address OUI as a classifier.

```
Switch(config)# macro auto mac-address-group OUI_PRINTER_PORT
oui list 0000AA
exit

Switch(config)# macro auto execute OUI_PRINTER_PORT  {
  if [[ $LINKUP -eq YES ]]
    then conf t
    interface $INTERFACE
    description OUI_PRINTER_PORT macro
    switchport
    switchport mode access
    switchport access vlan data_vlan
    power inline never
    spanning-tree portfast
    exit
  end
  fi

  if [[ $LINKUP -eq NO ]]
    then conf t
    interface $INTERFACE
    switchport access vlan data_vlan
    no spanning-tree portfast
    no description
    exit
  end
  fi
}
```

AutoSmart Ports are a great start to automating specific tasks when it comes to managing your campus LAN. It should be noted that even though AutoSmart Ports are not the most granular way to automate port configurations based on device, it is still a very powerful solution to help reduce some of the more arduous tasks that relate to day-to-day moves, additions, and changes (MACs).
Note For more information on AutoSmart Ports, please visit the following link: www.cisco.com/go/SmartOperations/

AutoConf

Similar to AutoSmart Ports, AutoConf is used to automate various functions within a Cisco Catalyst switch. However, unlike AutoSmart Ports, AutoConf is a template-based solution that is more granular and user friendly. Although these features accomplish similar outcomes, the configurations are applied in a different manner. Interface templates are configured and applied to a specific port or range of ports much like AutoSmart Ports. Table 7-2 lists some of the available predefined interface templates within a Cisco Catalyst switch.

Note AutoConf is available in IOS 15.2(2)E and IOS-XE 3.6 or later.

<table>
<thead>
<tr>
<th>Template Name</th>
<th>Template Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP_INTERFACE_TEMPLATE</td>
<td>Wireless access point interface template</td>
</tr>
<tr>
<td>DMP_INTERFACE_TEMPLATE</td>
<td>Digital media player interface template</td>
</tr>
<tr>
<td>IP_CAMERA_INTERFACE_TEMPLATE</td>
<td>IP camera interface template</td>
</tr>
<tr>
<td>IP_PHONE_INTERFACE_TEMPLATE</td>
<td>IP phone interface template</td>
</tr>
<tr>
<td>LAP_INTERFACE_TEMPLATE</td>
<td>Lightweight access point interface template</td>
</tr>
<tr>
<td>MSP_CAMERA_INTERFACE_TEMPLATE</td>
<td>Multiservices platform camera interface template</td>
</tr>
<tr>
<td>MSP_VC_INTERFACE_TEMPLATE</td>
<td>Multiservices platform VC interface template</td>
</tr>
<tr>
<td>PRINTER_INTERFACE_TEMPLATE</td>
<td>Printer interface template</td>
</tr>
<tr>
<td>ROUTER_INTERFACE_TEMPLATE</td>
<td>Router interface template</td>
</tr>
<tr>
<td>SWITCH_INTERFACE_TEMPLATE</td>
<td>Switch interface template</td>
</tr>
<tr>
<td>TP_INTERFACE_TEMPLATE</td>
<td>Telepresence interface template</td>
</tr>
</tbody>
</table>

Some of the key benefits of using templates are as follows:

- Simpler configuration and management than AutoSmart Port macros.
- All interface templates are customizable.
- Templates take up less room in the configuration file than AutoSmart Port macros.
- Template updates apply to all interfaces subscribing to the template.
- Templates can be per session or per port.
The following output shows an example of the built-in IP Phone template by issuing the `show template interface source built-in IP_PHONE_INTERFACE_TEMPLATE` command.

```
Switch# show template interface source built-in IP_PHONE_INTERFACE_TEMPLATE

Template Name       : IP_PHONE_INTERFACE_TEMPLATE
Modified            : No
Template Definition :
  spanning-tree portfast
  spanning-tree bpduguard enable
  switchport mode access
  switchport block unicast
  switchport port-security maximum 3
  switchport port-security maximum 2 vlan access
  switchport port-security violation restrict
  switchport port-security aging time 2
  switchport port-security aging type inactivity
  switchport port-security
  storm-control broadcast level pps 1k
  storm-control multicast level pps 2k
  storm-control action trap
  mls qos trust cos
  service-policy input AUTOCONF-SRND4-CISCOPHONE-POLICY
  ip dhcp snooping limit rate 15
  load-interval 30
  srr-queue bandwidth share 1 30 35 5
  priority-queue out
```

**Note** To see a list of all the built-in interface templates, issue the `show template interface source built-in all` command.

Below is a list of some of the common key points to keep in mind about AutoConf Templates:

- By default, all templates automatically use VLAN 1. This includes any access VLAN, voice VLAN, and native VLAN in regard to trunk ports.
- Templates applied to interfaces are not shown in running configuration. In order to see the configuration applied to an interface, issue the `show derived-config interface <interface>` command.
- EtherChannel interfaces do not support AutoConf interface templates.
- Once AutoConf is enabled globally, it is applied to all interfaces by default. To disable AutoConf on a per-interface basis, issue the `access-session inherit disable autoconf` command.
The template configuration itself does not show up in the running configuration unless the template is modified. For example, the access VLAN is changed from the default value of VLAN 1.

All template configuration settings applied to an interface are removed once the device is disconnected from the switch port.

### Enabling AutoConf on a Cisco Catalyst Switch

To enable AutoConf, the `autoconf enable` command must be issued from the global configuration mode. The following example illustrates the steps on how to enable AutoConf globally on a Cisco Catalyst Switch.

```text
Switch> enable
Switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# autoconf enable
Switch(config)# end
Switch#
```

AutoConf is now enabled globally on the Catalyst Switch. To verify AutoConf is working properly, a Cisco IP phone is connected into interface GigabitEthernet0/1 on the Catalyst switch. As displayed in the following output, once the phone is connected, AutoConf will apply the `IP_PHONE_INTERFACE_TEMPLATE` to the interface.

```text
Switch# show template binding target gigabitEthernet0/1

Interface Templates
===================
Interface: Gi0/1

<table>
<thead>
<tr>
<th>Method</th>
<th>Source</th>
<th>Template-Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic</td>
<td>Built-in</td>
<td>IP_PHONE_INTERFACE_TEMPLATE</td>
</tr>
</tbody>
</table>

Service Templates
=================
Interface: Gi0/1

<table>
<thead>
<tr>
<th>Session</th>
<th>Source</th>
<th>Template-Name</th>
</tr>
</thead>
</table>

Based on the previous output, the `IP_PHONE_INTERFACE_TEMPLATE` was successfully applied to the GigabitEthernet0/1 interface.
**Note** In general, to see the details of what settings are applied to an interface once a device is connected, issue the `show derived-config interface <interface_name>` command.

Notice that the applied template does not show up in the running configuration of the Catalyst switch. The following snippet shows the output of the `show running-config interface gigabitEthernet0/1` command, illustrating that the interface template is hidden in the running configuration.

```
Switch# show running-config interface gigabitEthernet0/1
Building configuration...

Current configuration : 36 bytes
!
interface GigabitEthernet0/1
end
```

To see the details of what settings were applied to the GigabitEthernet0/1 interface when the Cisco IP phone was connected, issue the `show derived-config interface gigabitEthernet0/1` command as shown in the following output.

```
Switch# show derived-config interface gigabitEthernet0/1
Building configuration...

Derived configuration : 669 bytes
!
interface GigabitEthernet0/1
  switchport mode access
  switchport block unicast
  switchport port-security maximum 3
  switchport port-security maximum 2 vlan access
  switchport port-security violation restrict
  switchport port-security aging time 2
  switchport port-security aging type inactivity
  switchport port-security
  load-interval 30
  srr-queue bandwidth share 1 30 35 5
  priority-queue out
  mls qos trust cos
  storm-control broadcast level pps 1k
  storm-control multicast level pps 2k
  storm-control action trap
  spanning-tree portfast
  spanning-tree bpduguard enable
```
service-policy input AUTOCONF-SRND4-CISCOPHONE-POLICY
ip dhcp snooping limit rate 15

Switch#

Modifying a Built-in Template

Commonly, built-in templates need to be modified to fit the desired configuration model of the environment. Modification of a built-in template allows for the flexibility of having a customized template, based on settings that align with the business needs. The following example lists the steps necessary to modify the built-in IP_PHONE_INTERFACE_TEMPLATE. These configuration steps will change the voice and data VLANs from the default of VLAN 1 to VLANs 11 and 13, respectively, and will add a custom description to the template.

Switch> enable
Switch# configure terminal
   Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# template IP_PHONE_INTERFACE_TEMPLATE
Switch(config-template)# switchport access vlan 11
Switch(config-template)# switchport voice vlan 13
Switch(config-template)# description CUSTOM_IP_PHONE_INTERFACE_TEMPLATE
Switch(config-template)# end
Switch#

To display the configuration changes made to the template, issue the show template interface source built-in IP_PHONE_INTERFACE_TEMPLATE command as shown in the following output.

Switch# show template interface source built-in IP_PHONE_INTERFACE_TEMPLATE
Building configuration...

Template Name       : IP_PHONE_INTERFACE_TEMPLATE
Modified            : Yes
Template Definition :
   spanning-tree portfast
   spanning-tree bpduguard enable
   switchport access vlan 11
   switchport mode access
   switchport block unicast
   switchport voice vlan 13
   switchport port-security maximum 3
   switchport port-security maximum 2 vlan access
   switchport port-security violation restrict
   switchport port-security aging time 2
   switchport port-security aging type inactivity
Once an AutoConf template has been modified, the template will now be visible in the running configuration of the Catalyst switch. The following snippet illustrates that the template is now present in the output of the `show running-config` command.

```plaintext
Switch# show running-config
Building configuration...

| Output omitted for brevity |

! autoconf enable
!

! template IP_PHONE_INTERFACE_TEMPLATE
  spanning-tree portfast
  spanning-tree bpduguard enable

switchport access vlan 11
  switchport mode access
  switchport block unicast

switchport voice vlan 13
  switchport port-security maximum 3
  switchport port-security maximum 2 vlan access
  switchport port-security violation restrict
  switchport port-security aging time 2
  switchport port-security aging type inactivity

switchport port-security
  storm-control broadcast level pps 1k
  storm-control multicast level pps 2k
  storm-control action trap
  mls qos trust cos
  service-policy input AUTOCONF-SRND4-CISCOPHONE-POLICY

ip dhcp snooping limit rate 15
load-interval 30
```

Switch#
description CUSTOM_IP_PHONE_INTERFACE_TEMPLATE
srr-queue bandwidth share 1 30 35 5
priority-queue out
!
! Output omitted for brevity

**Note** Even though the template is now visible in the running-config, it still does not list the configuration under the interface(s) that it is applied to.

Although the IP_PHONE_INTERFACE_TEMPLATE is modified and applied, the configuration is still hidden from the interface in the running-config. In order to see the customized configuration that is applied to the interface, the `show derived-config interface gigabitEthernet0/1` command must be used again. The following output shows the modified template that is applied to the gigabitEthernet0/1 interface.

Switch# show derived-config interface gigabitEthernet0/1
Building configuration...

! interface GigabitEthernet0/1
description CUSTOM_IP_PHONE_INTERFACE_TEMPLATE
switchport access vlan 11
switchport mode access
switchport block unicast
switchport voice vlan 13
switchport port-security maximum 3
switchport port-security maximum 2 vlan access
switchport port-security violation restrict
switchport port-security aging time 2
switchport port-security aging type inactivity
switchport port-security
load-interval 30
srr-queue bandwidth share 1 30 35 5
priority-queue out
mls qos trust cos
storm-control broadcast level pps 1k
storm-control multicast level pps 2k
storm-control action trap
spanning-tree portfast
spanning-tree bpduguard enable
service-policy input AUTOCONF-SRND4-CISCOPHONE-POLICY
ip dhcp snooping limit rate 15
end

Switch#
AutoConf is a feature that not only eases the burden of device management and configuration, it also allows for a zero-touch deployment model of commonly connected devices. AutoConf is often used in campus LANs as well as remote branch office deployments. Most organizations enforce a standard when it comes to the type of devices in their environment. Even though make, model, and form factors may differ, AutoConf can assist in reducing the manual configuration tasks needed to deploy different device types such as computers, printers, IP phones, IP cameras, and so forth. If a device supports both AutoConf and AutoSmart ports, it is recommended to use AutoConf first, then AutoSmart ports. However, using both features together could cause undesired results.


**Auto Security**

Cisco Auto Security is a feature that, when applied, automatically configures some of the most common baseline campus switching security features. Some of these features include:

- DHCP snooping
- Dynamic ARP inspection (DAI)
- Port Security

**DHCP Snooping** is a security feature that is designed to protect internally trusted DHCP servers and clients in your environment. DHCP Snooping works by verifying DHCP messages are received from only trusted DHCP servers within your campus environment. All messages from untrusted devices can be filtered or rate-limited, based on the desired configuration parameters. This security mechanism is to keep untrusted hosts from generating DHCP messages that could negatively impact your network. These DHCP messages can be malicious in nature or simply be the product of a misconfiguration. For example, a host computer has a DHCP server feature inadvertently turned on and is providing an unrouteable, incorrect IP address range to various devices in the environment. This will result in end hosts not being able to talk to the rest of the network. However, receiving a DHCP lease from any rogue server could be very problematic even if the IP address ranges are valid in your environment.

When enabled, the DHCP snooping feature keeps track of all devices sending and receiving DHCP messages. This information is stored in a table called the DHCP binding database. When DHCP messages are determined to be legitimate, they are processed normally. If for some reason the intercepted DHCP messages do not meet the proper criteria, the packets are discarded. This helps to protect your environment from DHCP snooping attacks.

**Dynamic ARP inspection (DAI)** is a feature that is used to prevent address resolution protocol (ARP) spoofing attacks. An ARP spoofing attack is when someone maliciously
injects a duplicate MAC address onto a LAN in an attempt to redirect traffic to an alternate destination. DAI uses the DHCP binding database to verify that there is a valid layer 2 MAC address to layer 3 IP address binding before allowing any traffic to be forwarded on the segment. If it is determined that there is not such a valid mapping, the invalid ARP packets are discarded.

Port Security is a security feature that protects the network by setting dynamic or hard MAC address limits on specific switch ports. For example, the following list provides some of the Port Security features that are available in Catalyst switches.

- Secure ports, based on statically assigned MAC addresses
- Secure ports, based on dynamically learned MAC addresses
- Limit dynamically learned MAC addresses—helps prevent CAM table flooding attacks
- Shut down port when violation occurs
- Restrict port and send SNMP trap when violation occurs

### Enabling Auto Security on a Cisco Catalyst Switch

The following example illustrates how to enable Auto Security on a Catalyst switch with a single command.

```
Switch> enable
Switch# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)# auto security
Switch(config)# end
```

To verify what interfaces the Auto Security configuration has been applied to, issue the `show auto security` command shown in the following output.

```
Switch# show auto security
Auto Security is Enabled globally

AutoSecurity is Enabled on below interface(s):
-----------------------------------------------
  GigabitEthernet0/1

Switch#
```

Because GigabitEthernet0/1 is configured as an access port, the following snippet illustrates the configuration that is visible in the running-config under that specific interface.

```
Switch# show running-config interface GigabitEthernet0/1
Building configuration...
```
In order to see the specific configuration that has been automatically applied to the Catalyst switch the `show auto security configuration` command must be issued. The following output depicts the steps necessary to verify the Auto Security configuration.

```
Switch# show auto security configuration
%AutoSecurity provides a single CLI config 'auto security'
to enable Base-line security Features like
DHCP snooping, ARP inspection and Port-Security

Auto Security CLIs applied globally:
---------------------------------------
ip dhcp snooping
ip dhcp snooping vlan 2-1005
no ip dhcp snooping information option
ip arp inspection vlan 2-1005
ip arp inspection validate src-mac dst-mac ip

Auto Security CLIs applied on Access Port:
----------------------------------------
switchport port-security maximum 2
switchport port-security maximum 1 vlan access
switchport port-security maximum 1 vlan voice
switchport port-security violation restrict
switchport port-security aging time 2
switchport port-security aging type inactivity
switchport port-security
ip arp inspection limit rate 100
ip dhcp snooping limit rate 100

Auto Security CLIs applied on Trunk Port:
--------------------------------------
ip dhcp snooping trust
ip arp inspection trust
switchport port-security maximum 100
```
As seen from the above configuration, Auto Security enables an entire baseline of security features on the Catalyst switch. All of these security features and settings have been streamlined into a single command. This automates the deployment of these features, which makes it easier to secure the campus LAN environment.

**Note** Although many First-Hop Security features have been available in various IOS versions for some time, the Auto Security feature is available in IOS XE 3.6.0E and IOS 15.2(2)E and later.


### Quality of Service for Campus Architectures

Quality of Service (QoS) is an integral part of any campus environment. QoS allows for the prioritization of specific traffic flows as they traverse over the campus network. For example, it may be desirable to allow voice and video traffic to have priority over bulk FTP traffic during a time of network congestion. One of the most common reasons that QoS is not deployed is due to its complexity. This section will discuss some different ways to automate the deployment of QoS for LAN devices.

**Note** A base understanding of QoS is assumed. QoS fundamentals are not covered in this chapter. To become more familiar with QoS and its components, please visit: [www.cisco.com/go/qos](http://www.cisco.com/go/qos)

### AutoQoS on Campus LAN Devices

As campus networks continue to grow, more emphasis is being put on the LAN. Today, it is becoming even more important to capitalize on the available LAN bandwidth as much as possible. Often, campus networks are designed with a specific set of goals in mind. For example, the following list are some of the more common business drivers and use cases that put demand on the campus LAN infrastructure:

- Gigabit Ethernet to the desktop
- Campus video communications
- Voice and IP phones
Alternatively, there are some other use cases that are beginning to be more prevalent in enterprise networks. These different, but not uncommon use cases are increasing the demand for connectivity in the LAN:

- Wayfinding devices
- Digital signage
- HVAC systems
- Manufacturing/industrial networks
- Building lighting

All of the above use cases are putting increased demand on the network and, by default, demand on the network engineering team.

**Enabling AutoQoS on a Cisco Catalyst Switch**

To enable AutoQoS, the following configuration steps must be followed:

**Step 1.** Enable AutoQoS globally

**Step 2.** Enable AutoQoS settings under interface

AutoQoS is enabled globally in the following example on the Catalyst switch by issuing the `auto qos global compact` command from the global configuration prompt. Once the feature is enabled globally, it can be verified with the `show auto qos` command.

Switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# auto qos global compact
Switch(config)# end
Switch# show auto qos
AutoQoS not enabled on any interface

Switch#

As you can see from the output of the `show auto qos` command in the following code snippet, there are no interfaces currently configured with any AutoQoS parameters. Once AutoQoS is enabled globally, you must then specify the interface configuration settings. For example, see the following output that illustrates how to enable the AutoQoS settings under a Gigabit Ethernet interface of a Catalyst switch. The configuration shown is for a Cisco IP phone.

Switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# interface GigabitEthernet0/1
Switch(config-if)# auto qos voip cisco-phone
Switch(config-if)# end
Switch#

Now that AutoQoS is enabled globally and there is an interface with AutoQoS settings applied to it, the `show auto qos` command is re-issued to verify the configuration as shown in the following snippet. Based on the output of the `show auto qos` command, we see that there is a difference in the information displayed as opposed to output shown previously. When AutoQoS is enabled under the GigabitEthernet0/1 interface, it now includes the interface configuration in the show command.

Switch# show auto qos
GigabitEthernet0/1
auto qos voip cisco-phone
Switch#

In order to display the actual QoS settings that get applied to the GigabitEthernet0/1 interface when a Cisco IP phone is connected, the `show auto qos interface GigabitEthernet0/1 configuration` command must be issued. The following snippet shows that based on the output of this command, there is an ingress policy named AUTOQOS-PPM-SRND4-CISCOPHONE-POLICY that is applied to the GigabitEthernet0/1 interface. The output also shows that the outbound egress priority queue is enabled and that the interface has been set to automatically trust the DSCP markings from the Cisco IP phone.

Switch# show auto qos interface GigabitEthernet0/1 configuration
GigabitEthernet0/1
auto qos voip cisco-phone
Ingress Policy: AUTOQOS-PPM-SRND4-CISCOPHONE-POLICY
Egress Priority Queue: enabled
The port is mapped to qset : 1
Trust device: cisco-phone

Next, to further validate the settings within the AUTOQOS-PPM-SRND4-CISCOPHONE-POLICY that is applied to the GigabitEthernet0/1 interface, we issue the show policy-map AUTOQOS-PPM-SRND4-CISCOPHONE-POLICY command as shown in the following output.

Switch# show policy-map AUTOQOS-PPM-SRND4-CISCOPHONE-POLICY
Policy Map AUTOQOS-PPM-SRND4-CISCOPHONE-POLICY
Class AUTOQOS_PPM_VOIP_DATA_CLASS
    set dscp ef
    police 128000 8000 exceed-action policed-dscp-transmit
Class AUTOQOS_PPM_VOIP_SIGNAL_CLASS
    set dscp cs3
    police 32000 8000 exceed-action policed-dscp-transmit
Based on the previous output, we can see that the following parameters have been set in the QoS policy-map applied to the GigabitEthernet0/1 interface on the Catalyst switch:

- Voice data packets are being marked with the DSCP value of EF (46)
- Policing of the VOIP_DATA_CLASS is set to 128Kbps
- Call signaling packets are being marked with the DSCP value of CS3
- Policing of the VOIP_SIGNAL_CLASS is set to 32Kbps
- All other packets are being marked with DSCP value of DEF AULT (0)
- Policing of the DEFAULT_CLASS is set to 10Mbps

The following snippet illustrates the output of the `show auto qos voip cisco-phone configuration` command, which is an alternate way of displaying the AutoQoS configuration that will be applied to an interface when a Cisco IP phone is connected. This command will also display the DSCP/CoS markings, queuing strategy, and associated thresholds settings that will be applied.

```
Switch# show auto qos cisco-phone configuration

Traffic(DSCP / COS)         IngressQ-Threshold    EgressQ-Threshold
---------------------------------------------------------------------
VoIP(46/5)                         N/A - N/A             01 - 01
Signaling(24/3)                    N/A - N/A             03 - 01
Best-Effort(00/0)                  N/A - N/A             02 - 01
```

All of the QoS settings mentioned above were deployed by issuing only two commands: the `auto qos global compact` global command and the `auto qos voip cisco-phone` interface command. We can begin to see how powerful tools like AutoQoS can be in a campus environment, especially with hundreds to thousands of connected host devices. The following section of this chapter will cover deploying AutoQoS in the campus WAN environment.

**AutoQoS on Campus WAN Devices**

The best practice in general from a QoS perspective is to mark the traffic closest to the source and carry those markings across your LAN and WAN end-to-end. The biggest reason for this is so that end users and applications have a consistent experience. Marking and prioritizing traffic on the LAN is just one step in a bigger QoS design. Using AutoQoS for the WAN, you can simplify the steps needed to achieve that end-to-end user and application experience. Figure 7-2 illustrates the high level end-to-end QoS design model from an IP phone in one location to an IP phone in another location.
Note Although we will not discuss AutoQoS for WAN in depth in this chapter, the purpose of this section is to inform the readers that there are tools for AutoQoS on Cisco routers.

Figure 7-2 End-to-end QoS example

As you can see based on Figure 7-2, the QoS markings are kept intact from source to destination across the campus LAN and WAN networks. In this specific case, voice data traffic from Phone-1 to Phone-2 is marked with DSCP EF (46), and those markings are honored on a hop-by-hop basis across the entire network. This is called per-hop behavior (PHB).

Enabling AutoQoS on a Cisco ISR Router

The following example lists the steps that are necessary to enable AutoQoS for the WAN on a Cisco ISR router.

```
Router# configure terminal
Router(config)# interface FastEthernet0/1
Router(config-if)# auto qos voip
Router(config-if)# end
Router#
```

One of the convenient things about AutoQoS for the WAN is that by enabling it on one of the interfaces of the router, it automatically enables the feature globally. Furthermore, it applies all the QoS policy-maps and other settings automatically. The following snippet illustrates an example output of the `show auto qos` command from a Cisco ISR router, illustrating what features AutoQoS will automatically activate when the feature is enabled.

```
Router# show auto qos
!
 policy-map AutoQoS-Policy-UnTrust
 class AutoQoS-VoIP-RTP-UnTrust
   priority percent 70
   set dscp ef
 class AutoQoS-VoIP-Control-UnTrust
   bandwidth percent 5
```
```
set dscp af31
class AutoQoS-VoIP-Remark
    set dscp default
    class class-default
        fair-queue
    
    class-map match-any AutoQoS-VoIP-Remark
        match ip dscp ef
        match ip dscp cs3
        match ip dscp af31
    
    class-map match-any AutoQoS-VoIP-Control-UnTrust
        match access-group name AutoQoS-VoIP-Control
    
    class-map match-any AutoQoS-VoIP-RTP-UnTrust
        match protocol rtp audio
        match access-group name AutoQoS-VoIP-RTCP
    
    ip access-list extended AutoQoS-VoIP-RTCP
        permit udp any any range 16384 32767 (6 matches)
    
    ip access-list extended AutoQoS-VoIP-Control
        permit tcp any any eq 1720
        permit tcp any any range 11000 11999
        permit udp any any eq 2427
        permit tcp any any eq 2428
        permit tcp any any range 2000 2002
        permit udp any any eq 1719
        permit udp any any eq 5060
    
    rmon event 33333 log trap AutoQoS description "AutoQoS SNMP traps for Voice Drops" owner AutoQoS
    rmon alarm 33333 cbQosCMDropBitRate.34.14175073 30 absolute rising-threshold 1 33333 falling-threshold 0 owner AutoQoS

    FastEthernet0/1 -
        
        interface FastEthernet0/1
            service-policy output AutoQoS-Policy-UnTrust
```

**Note**: AutoQoS for the WAN is platform dependent. To learn more about AutoQoS for WAN and what platforms the feature is supported on, please visit: http://www.cisco.com/c/en/us/products/ios-nx-os-software/autoqos/index.html
AutoQoS, in conjunction with some of the other automation mechanisms discussed earlier in the Automatic Port Profiling section of this chapter, can start to build a very robust and powerful tool set. This tool set can help network engineers ease the operational complexity of managing a constantly changing campus network environment. Chapter 8 “Network Automation Tools for Campus Environments” will highlight another tool set known as the application policy infrastructure controller enterprise module (APIC-EM). APIC-EM offers a wide variety of features that include tools to assist in configuring and automating quality of service in campus environments. We will also discuss some future APIC-EM applications.

**Automating Management and Monitoring Tasks**

This section will discuss a very robust set of tools that are built-in to many Cisco devices such as:

- Smart Call Home
- Tcl Shell
- Embedded Event Manager (EEM)

These tools are designed to make life a bit easier for the network operations staff by leveraging on-box automation.

**Smart Call Home**

Cisco’s Smart Call Home is a feature that is built into a large number of Cisco devices that allows the devices to automatically reach out to Cisco TAC when there is an issue in your campus environment. Smart Call Home can report a wide variety of different events. For example:

- Generic online diagnostics (GOLD)
- Syslog events
- Environment events and alarms
- Inventory and configuration
- Field notices
- Product security incident response team (PSIRT) notifications

There are three primary ways that Smart Call Home can collect this information from the IOS: Alert Groups and Profiles, collecting show commands, and interaction with the CLI. This information is sent via one of three different transport modes: HTTP(S) direct, HTTP(S) via a transport gateway, or via email through a transport gateway. A transport gateway is a device that securely forwards Call Home messages that are sourced from devices within the network. The information that is gathered and sent to Cisco TAC is then stored in a database within Cisco’s data centers. Once the information is collected
Automating Management and Monitoring Tasks

and stored in the database, you will be able to view the information from a web portal where you can manage all your devices. Smart Call Home allows TAC to do multiple things with the collected information:

- Automatically create TAC service requests, based on issues with the device(s)
- Notify the Cisco partner should they need to be contacted
- Notify the device owner that there is something going on with the device(s)

This helps make your business more proactive, rather than reactive. An example of Smart Call Home would be if you have a Catalyst 4500 series switch and one of the power supplies failed in the middle of the night. Instead of having to wake up, open a TAC case, and upload the serial number of the switch and the configuration and go through troubleshooting steps, the switch would have used Smart Call Home to contact TAC and upload all the necessary information and a TAC case would have already been opened automatically. In turn, an RMA could be issued automatically for the failed part. This drastically reduces the amount of time and effort engineers have to spend, going through the motions of all the steps mentioned above in order to get a replacement power supply and bring the network back to 100 percent. In addition to this, there is an anonymous reporting feature that allows Cisco to receive minimal error and health information from various devices.

There are six basic steps to enable Cisco’s Smart Call Home feature. Those steps are as follows:

- Enable Call Home
- Configure contact email address
- Activate CiscoTAC-1 profile
- Set transport mode
- Install security certificate
- Send a Call Home inventory to start the registration process

**Enabling Smart Call Home on an Cisco Catalyst Switch**

The following example depicts the process for setting up Smart Call Home on a Catalyst switch.

```
Switch# configure terminal
Switch(config)# service call-home
Switch(config)# call-home
Switch(cfg-call-home)# contact-email-addr neteng@yourcompany.com
Switch(cfg-call-home)# profile CiscoTAC-1
Switch(cfg-call-home-profile)# active
Switch(cfg-call-home-profile)# destination transport-method http
Switch(cfg-call-home-profile)# exit
```
Switch(config-call-home)# exit
Switch(config)# crypto pki trustpoint cisco
Switch(ca-trustpoint)# enrollment terminal
Switch(ca-trustpoint)# revocation-check crl none
Switch(ca-trustpoint)# exit
Switch(config)# crypto pki authenticate cisco

Enter the base 64 encoded CA certificate.
End with a blank line or the word "quit" on a line by itself

-----BEGIN CERTIFICATE-----
MIICPDCCAaUCEDyRMcsf9tAbDpq40BS/Er4wDQYJKoZIhvcNAQEBBFQAwXrELMAkGA1UEBMCV
MxSezA8B9NVBAotD1lcm1TaWdlLCB3b3luMzRtc2VjcmV0czAdBgNVHRMBAf8E
MIIE0DCCBDmgAwIBAgIQJQzo4DBhLp8rifcFTXz4/TANBgkqhkiG9w0BAQUFADBfMQswCQYDVQQGEwJ
VUzEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xNzA1BgNVBAsTLkNsYXNzIDMgUHVibGljIFByaW1h
cnkgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkwHhcNMDYxMTA4MDAwMDAwWhcNMjExMTA3MjM1OTU5WjC
LCAwHQYDVQQLEzEoYykgMjAwNiBWZXJpU2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSB0
b3JpdGlvbiBGa1UdIARaC50MCwGCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1nm
AMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XrrcxiV0i6CqppkKzj/i5Vbexe0uz/o9+B1f70Pb
3MlVCygDy3vju3jw2IIIPVf76onKWS5F0uUrjxue6/WkkcIzSdhD2yppS9XKP
6HRTGdljAxVhcPasz3Bj0Z2t3b1bT1r8Vd6w9K1l8g8ckmcy5FfQBO+QueQASb66n4z/Ar
cP07gi+s3l+2016zy9v9ar911kTMZHERx3yQgSGT2RT+cCpSx4/VBEnkjWNHiDxrgvV+8
R7r0fk/PlaOnnDRT8Q8cnuMCUI71P59zuDMKz10/NIeWiuZ6CVAgMBAAGggeg508n1lBz
APhB9VHVMBMAf#ETBADAQ/MDExGAIudWQgMcwJgAoKoCKG1B0dBA6Sy9jcmwmdVys
Xnp
Z24uY29tL3BjYy5M5gpA4GAIudWw/SQQAw1BBjA9B9gIVNSAENjA0MDIGBFUdIAAwXk
AoB9g9g9FBQOEaCARGCyeH0ckm6M6y93j3cumdVysXnpZ24uY29tL2NwresarB9gIVNSA4QFgQU
f9Nlpe8dL7LmK4DAnz2zmNagQeMTMwWQYKWBQtWQUAqWETT8oV2gWh2ZMpcVRYfA1h
To obtain the proper certificate to paste into the call configuration, please visit the following link to get the Smart Call Home user guide for your model of equipment:

Once you complete the certificate import process, you must then initiate a call home to begin the registration process for the device. Before we begin the call home process, we will enable the `debug event manager action cli` command as the following snippet depicts. This will show the steps that the call-home feature is taking. It is important to remember that call-home uses embedded event manager (EEM) to function. The following example also shows the `call-home` command that is used to initiate the call-home and registration process on a Cisco Catalyst switch.

```
Switch# debug event manager action cli
Debug EEM action cli debugging is on
Switch# call-home send alert-group inventory profile CiscoTAC-1
Sending inventory info call-home message ...
Please wait. This may take some time ...
```
Switch#
Dec  7 22:48:38.120: %HA_EM-6-LOG: callhome : DEBUG(cli_lib) : : OUT : ROM:
Dec  7 22:48:38.120: %HA_EM-6-LOG: callhome : DEBUG(cli_lib) : : OUT : Last reload reason: power-on
Dec  7 22:48:38.120: %HA_EM-6-LOG: callhome : DEBUG(cli_lib) : : OUT: This product contains cryptographic features and is subject to United States and local country laws governing import, export, transfer and use. Delivery of Cisco cryptographic products does not imply third-party authority to import, export, distribute or use encryption.
Dec  7 22:48:38.120: %HA_EM-6-LOG: callhome : DEBUG(cli_lib) : : CTL : 20+ lines read from cli, debug output truncated
Now that this step is complete, an email will be sent to the email address used in the CiscoTAC-1 profile as shown in Figure 7.3. In this case, that email address is neteng@yourcompany.com. Once that email is received, to complete the registration process you must follow the directions in the email. You must also have a valid contract associated to the device you are trying to register to the Smart Call Home portal. Following the link will redirect you to the Smart Call Home Web Portal as shown in Figure 7-4. Once logged into the portal, the device registration process can be completed.

Figure 7-3  Email from Cisco Smart Call Home Tool
To verify that Smart Call Home is running on your device, issue the `show call-home` command from the privileged exec prompt. The following snippet displays the output from the `show call-home` command on a Cisco Catalyst Switch. There are many different options that can be configured with Smart Call Home. The following alert groups are enabled automatically when configuring Smart Call Home with the `call-home send alert-group inventory profile CiscoTAC-1` command:

- Configuration
- Diagnostic
- Environment
- Inventory
- Syslog

```
Switch# show call-home
Current call home settings:
call home feature : enable
call home message's from address: Not yet set up
call home message's reply-to address: Not yet set up
vrf for call-home messages: Not yet set up
contact person's email address: neteng@yourcompany.com
contact person's phone number: Not yet set up
street address: Not yet set up
customer ID: Not yet set up
contract ID: Not yet set up
site ID: Not yet set up
```
source ip address: Not yet set up
source interface: Not yet set up
Mail-server: Not yet set up
Rate-limit: 20 message(s) per minute

Available alert groups:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Enable</td>
<td>configuration info</td>
</tr>
<tr>
<td>diagnostic</td>
<td>Enable</td>
<td>diagnostic info</td>
</tr>
<tr>
<td>environment</td>
<td>Enable</td>
<td>environmental info</td>
</tr>
<tr>
<td>inventory</td>
<td>Enable</td>
<td>inventory info</td>
</tr>
<tr>
<td>syslog</td>
<td>Enable</td>
<td>syslog info</td>
</tr>
</tbody>
</table>

Profiles:

Profile Name: CiscoTAC-1

Switch#

**Note** For more information on Smart Call Home, please visit: https://supportforums.cisco.com/community/4816/smart-call-home

**Tcl Shell**

Tcl Shell is a feature that is built into Cisco routers and switches that allows engineers to interact directly with the device by using various Tcl scripts. Tcl scripting has been around for quite some time and is a very useful scripting language. Tcl provides many ways to streamline different tasks that can help with day-to-day operations and monitoring of a network. Some of the following are tasks that can be automated by using these scripts:

- Verify IP and IPv6 reachability, using ping
- Verify IP and IPv6 reachability, using Traceroute
- Check interface statistics
- Retrieve SNMP information by accessing MIBs
- Send email messages containing CLI outputs from Tcl scripts

Most often, basic Tcl scripts are entered line by line within the Tcl shell, although, for some of the more advanced scripting methods, you can load the script into the flash of the device you are working on and execute the script from there. These scripts have to be in a specific Tcl format as shown in the following examples. The following example illustrates how to enter the Tcl shell on a Cisco router and execute a simple ping script.
Router# tclsh
Router(tcl)# foreach address {
  >tcl)# 192.168.0.2
  >tcl)# 192.168.0.3
  >tcl)# 192.168.0.4
  >tcl)# 192.168.0.5
  >tcl)# 192.168.0.6
  >tcl)# } { ping $address
  >tcl)# }
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.4, timeout is 2 seconds:
...
Success rate is 0 percent (0/5)
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/4 ms
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/4 ms
Router(tcl)# tclquit
Router#

An alternate to entering the DNS node names or IP addresses in a line-by-line fashion, you can also enter some of the script commands on a single line within the Tcl shell. For instance, the following example shows a similar ping script to the one entered before, but now it is executed on the same line within the Tcl shell.

Router# tclsh
Router(tcl)# foreach address [192.168.0.2 192.168.0.3 192.168.0.4] { ping $address}
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.4, timeout is 2 seconds:
...
Success rate is 0 percent (0/5)
Router(tcl)# tclquit
Router#

To abort a ping that is timing out while running a script, press and hold the
CTRL+Shift keys and press the 6 key for each failing ping, then release all keys. This
speeds up the script to keep processing past the node(s) that are not responding and does
not stop the script from running.

To execute Tcl Scripts from the local flash memory, you would need to store the script
in flash and then call the script by file name. Scripts can be stored on the device’s local
flash, USB flash, or compact flash. Tcl scripts can be transferred into the IOS File System
(IFS) by using SCP, TFTP, FTP, or RCP. From a security perspective, SCP is preferred due
to its use of SSH. To execute a locally stored script, the source command from within the
Tcl shell prompt can be used. The following example illustrates the steps to call a script
named ping.tcl from the local flash on a device. This script is an example of the same
ping script that was shown earlier in this chapter.

Note The scripts that are stored locally to the device should be named in the following
manner: filename.tcl

Router# tclsh
Router(tcl)# source flash:ping.tcl

Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
Type escape sequence to abort.
Sending 5, 64-byte ICMP Echos to 192.168.0.4, timeout is 2 seconds:
...
Success rate is 0 percent (0/5)
Router(tcl)# tclquit
Router#
Embedded Event Manager (EEM)

Embedded Event Manager (EEM) is a very flexible and powerful tool within Cisco IOS. EEM allows engineers to build software applets that can automate many tasks. EEM also derives some of its power from the fact that you can build custom scripts using Tcl so that they automatically execute, based on the output of an action or an event on a device. One of the main benefits of EEM is that it is all contained within the local device. There is no need to rely on an external scripting engine or monitoring device in most cases. Figure 7-5 illustrates some of the event detectors and how they interact with the IOS subsystem.

Figure 7-5  EEM Event Detectors

EEM Applets

EEM applets are comprised of multiple building blocks. In this chapter, we will focus on the two of the primary building blocks that make up EEM applets. Those building blocks are called events and actions. These EEM applets use a similar logic to the if-then statements found in some of the more common programming languages. For instance, if an event happens, then an action is taken. In the following example, we illustrate a very common EEM applet that is monitoring syslog messages on a router. This particular
applet is looking for a specific syslog message, stating that the Loopback0 interface went down. The specific syslog message is matched using regular expressions. This is a very powerful and granular way of matching patterns. If this specific syslog pattern is matched (an event) at least once, then the following actions will be taken:

- The Loopback0 interface will be shutdown and brought back up (shutdown, then no shutdown)
- The router will generate a syslog message that says “I’ve fallen, and I can’t get up!”
- An email message will be sent to the network administrator that includes the output of the show interface loopback0 command.

```
event manager applet LOOP0
  event syslog pattern "Interface Loopback0.* down" period 1
  action 1.0 cli command "enable"
  action 2.0 cli command "config terminal"
  action 3.0 cli command "interface loopback0"
  action 4.0 cli command "shutdown"
  action 5.0 cli command "no shutdown"
  action 5.5 cli command "show interface loopback0"
  action 6.0 syslog msg "I've fallen, and I can't get up!"
  action 7.0 mail server 10.0.0.25 to neteng@yourcompany.com from no-reply@yourcompany.com subject "Loopback0 Issues!" body "The Loopback0 interface was bounced. Please monitor accordingly. "$cli_result"
```

**Note** Remember to include the `enable` and `configure terminal` commands at the beginning of actions within your applet. This is necessary as the applet assumes you are in exec mode, not privileged exec or config mode. In addition, if you are using AAA command authorization, you will want to include the `event manager session cli username username` command. Otherwise, the CLI commands in the applet will fail. It is also good practice to use decimal labels similar to 1.0, 2.0, and so forth when building applets. This allows you to insert and action between other actions in the future. For example, 1.5 will allow you to insert an action between 1.0 and 2.0. Remember that labels are parsed as strings, which means 10.0 would come after 1.0, not 9.0.

Based on the output from the `debug event manager action cli`, you can see the actual actions taking place when the applet is running. The following example shows the applet being engaged when we issue the `shutdown` command on the Loopback0 interface. It also shows that there was an error when trying to connect to the SMTP server to send the email to the administrator. This is because the actual SMTP server we are using for this test is not configured. Notice that because we used the `$ cli_result` keyword in the configuration, it will include the output of any CLI commands that were issued in the applet. In this case, the output of the `show interface Loopback0` command will be included in the debug and the mail message.
Switch# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Switch(config)# interface loopback0
Switch(config-if)# shutdown
Switch(config-if)#
Dec  6 17:21:59.214: %LINK-5-CHANGED: Interface Loopback0, changed state to administratively down
Dec  6 17:21:59.221: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : Switch>
Dec  6 17:21:59.221: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : IN : Switch>enable
Dec  6 17:21:59.231: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : Switch#
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : Loopback0 is administratively down, line protocol is down
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: Hardware is Loopback
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: MTU 1514 bytes, BW 800000 Kbit/sec, DLY 5000 usec,
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: Encapsulation LOOPBACK, loopback not set
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: Keepalive set (10 sec)
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: Last input never, output never, output hang never
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: Last clearing of "show interface" counters never
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: Output queue: 0/0 (size/max)
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: 5 minute input rate 0 bits/sec, 0 packets/sec
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: 5 minute output rate 0 bits/sec, 0 packets/sec
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: 0 packets input, 0 bytes, 0 no buffer
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: Received 0 broadcasts (0 IP multicasts)
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: 0 runts, 0 giants, 0 throttles
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: 0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
Dec  6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT: 0 packets output, 0 bytes, 0 underruns
Dec 6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : 0 output errors, 0 collisions, 0 interface resets
Dec 6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : 0 unknown protocol drops
Dec 6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : CTL : 20+ lines read from cli, debug output truncated
Dec 6 17:21:59.252: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : IN : Switch#config terminal
Dec 6 17:21:59.266: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : Enter configuration commands, one per line. End with CTRL/Z.
Dec 6 17:21:59.266: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : Switch(config)#
Dec 6 17:21:59.266: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : IN : Switch(config)#interface loopback0
Dec 6 17:21:59.277: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : Switch(config-if)#
Dec 6 17:21:59.277: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : IN : Switch(config-if)#shutdown
Dec 6 17:21:59.287: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : Switch(config-if)#
Dec 6 17:21:59.287: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : IN : Switch(config-if)#no shutdown
Dec 6 17:21:59.298: %HA_EM-6-LOG: LOOP0 : DEBUG(cli_lib) : : OUT : Switch(config-if)#
Dec 6 17:21:59.298: %HA_EM-6-LOG: LOOP0: I've fallen and I can't get up!
Dec 6 17:22:11.314: %HA_EM-6-LOG: LOOP0: I've fallen and I can't get up!
Note For troubleshooting purposes, using the debug event manager all command will show all the outputs for the configured actions while the applet is being executed. For instance, it will show the same output as shown above but will include more details on all the other actions. To specifically troubleshoot the mail configuration and related error messages in an EEM Applet, the debug event manager action mail command is most useful as it filters out all the other unnecessary debug messages while you are trying to troubleshoot the mail configuration. This will allow you to focus on SMTP errors as shown in the previous example.
Another very useful aspect of EEM applets is that CLI patterns can be matched as an event. This means that when certain commands are entered into the router via CLI, they can trigger an EEM event within an applet. Then the configured actions will take place as a result of the CLI pattern being matched. The following example uses another common
EEM applet to match the CLI pattern “wr mem”. Once the applet is triggered, the following actions will be invoked:

- The router will generate a syslog message that says “Configuration File Changed!”
- The startup-config will be copied to a TFTP server.
- Generate a syslog message stating that the configuration has been successfully saved.

```plaintext
event manager environment filename Router.cfg
event manager environment tftpserver tftp://10.1.200.29/
event manager applet BACKUP-CONFIG
event cli pattern "write mem.*" sync yes
action 1.0 cli command "enable"
action 2.0 cli command "configure terminal"
action 3.0 cli command "file prompt quiet"
action 4.0 cli command "end"
action 5.0 cli command "copy start $tftpserver$filename"
action 6.0 cli command "configure terminal"
action 7.0 cli command "no file prompt quiet"
action 8.0 syslog priority informational msg "Configuration File Changed! TFTP backup successful."
```

**Note** The `file prompt quiet` command disables the IOS confirmation mechanism that asks you to confirm your actions.

**Note** The priority and facility of the Syslog messages can be changed to fit your environment’s alerting structure. For example, we used informational in the previous example.

As seen in the previous examples there are multiple ways to call out specific EEM environment values. The first example illustrated that you can use a single line to configure the mail environment and send messages with CLI output results. Using the event manager environment variables shown in the second example, you can statically set different settings that you can call on from multiple actions instead of calling them out individually on a single line. Although you can create custom names and values that are arbitrary and can be set to anything, it is good practice to use common and descriptive variables. Table 7-3 lists some of the most commonly used email variables in EEM.
Table 7-3  Common EEM Email Variables

<table>
<thead>
<tr>
<th>EEM Variable</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>_email_server</td>
<td>SMTP server IP address or DNS name</td>
<td>10.0.0.25 or MAILSVR01</td>
</tr>
<tr>
<td>_email_to</td>
<td>Email address to send email to</td>
<td><a href="mailto:neteng@yourcompany.com">neteng@yourcompany.com</a></td>
</tr>
<tr>
<td>_email_from</td>
<td>Email address of sending party</td>
<td><a href="mailto:no-reply@yourcompany.com">no-reply@yourcompany.com</a></td>
</tr>
<tr>
<td>_email_cc</td>
<td>Email address of additional email receivers</td>
<td><a href="mailto:helpdesk@yourcompany.com">helpdesk@yourcompany.com</a></td>
</tr>
</tbody>
</table>

EEM and Tcl Scripts

Using an EEM applet to call Tcl scripts is another very powerful aspect of EEM. We have covered multiple ways to use EEM applets. In this section, we will discuss how to call a Tcl script from an EEM applet. The previous sections on EEM showed multiple ways of executing actions, based on the automatic detection of specific events when they are happening. This example shows how to manually execute an EEM applet that will, in turn, execute a Tcl script that is locally stored in the device's flash memory.

It is important to understand that there are many different ways to use EEM and that manually triggered applets are also a very useful tool. The following example depicts an EEM script that is configured with the event none command. This means that there is no automatic event that the applet is monitoring and that this applet will only run when it is triggered manually. To manually run an EEM applet, the event manager run command must be used as illustrated in second output.

```
event manager applet Ping
  event none
  action 1.0 cli command "enable"
  action 1.1 cli command "tclsh flash:/ping.tcl"

Router# event manager run Ping

Dec 6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Sending 5, 100-byte ICMP Echos to 192.168.0.2, timeout is 2 seconds:
Dec 6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/4 ms
```
Dec  6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Sending 5, 100-byte ICMP Echos to 192.168.0.3, timeout is 2 seconds:
Dec  6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms
Dec  6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Sending 5, 100-byte ICMP Echos to 192.168.0.4, timeout is 2 seconds:
Dec  6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/3 ms
Dec  6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Sending 5, 100-byte ICMP Echos to 192.168.0.5, timeout is 2 seconds:
Dec  6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/4 ms
Dec  6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Sending 5, 100-byte ICMP Echos to 192.168.0.6, timeout is 2 seconds:
Dec  6 19:32:16.711: %HA_EM-6-LOG: Ping : DEBUG(cli_lib) : : OUT : Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms

For reference, see the following snippet for the exact content of the ping.tcl script used in the manually triggered EEM applet in the previous example. To see the contents of a TCL script that resides in flash, issue the `more` command followed by the file location and filename. The `more` command can be used to view all other text based files stored in the local flash as well.

```
Router# more flash:ping.tcl
foreach address {
  192.168.0.2
  192.168.0.3
  192.168.0.4
  192.168.0.5
  192.168.0.6
} { ping $address}
```
EEM Summary

There are many ways to utilize EEM. From applets to scripting, the possibly use cases can only be limited by the engineer's imagination. EEM provides on-box monitoring of various different components based on a series of events. Once an event is detected, an action can take place. This helps make some of the network monitoring more proactive, rather than reactive. This can also reduce the load on the network and improve efficiency from the monitoring system because now the devices can simply report when there is something wrong instead of continually asking the devices if there is anything wrong.


Summary

By automating daily configuration tasks, you gain some of the following benefits:

- Increased agility
- Reduced Opex
- Lower overall TCO
- Streamlined management
- Reduction of human error
- Increased visibility

Keeping the above in mind, then adding the fact that many organizations are dealing with lean IT problems and high turnover, network engineers are being asked to do more with less. Utilizing some of the tools that were covered in this chapter can help alleviate some of the pressure put on IT staff by offloading some of the more tedious, time-consuming, and repetitious tasks. This will allow the network engineer to focus more on critical mission responsibilities like network design and growth planning.
Symbols

* (asterisk), 32
\ (backslash), 90
^ (caret), 32
./ construct (bash), 90
$ (dollar sign), 32
= (equal sign), 92
< (left arrow), 92
. (period), 32, 33
+ (plus sign), 32
# (pound sign), 29
>> redirector, 94
> (right arrow), 92
[] (square brackets), 32, 33
16 nanometer ASIC, 68

A

aaaLogin method, 185
aaaLogout method, 186
aaaRefresh method, 185
aborting pings, 245

Access Control Lists (ACLs)
APIC-EM Policy application, 286–287
overview, 79
Access module (Cobra SDK), 200
accessing
dictionaries, 28
lists, 27
virtual environments, 163–164
ACI (Application Centric Infrastructure).
See Cisco
ACI (Application Centric Infrastructure)
ACIs (Access Control Lists)
APIC-EM Policy application, 286–287
benefits of, 79
actions (EEM), 112–113
activate command, 163–164
addresses (IP), 130–131
ALE (application leaf engine), 67
ANPs (application network profiles), 72
Ansible
Nexus 9000, 157–158
overview, 156–157
API inspector (APIC), 179–182
APIC REST Python Adapter (Arya)
AryaLogger, 207–208
installing, 207–208
WebArya, 211
APIC RESTful API
API inspector, 179–182
APIC automation with UCS Director, 211
APIC configuration automation, 188–191
atomic mode, 174–175
authentication
authentication methods, 185–186
overview, 182–186
with Python, 186–188
Cobra SDK
Arya (APIC REST Python Adapter), 207–211
authentication, 201
documentation, 198
installing, 199–200
libraries, 200–201
modules, 200
objects, 202–204
tenant configuration example, 204–207
components, 175–176
event subscription, 196–198
forgiving mode, 174–175
GUI, 178
invoking, 176–178
object save-as, 178–179
programmability, 192–196
read operations, 176
Visore, 182–185
write operations, 177
APIC-EM (application policy infrastructure controller enterprise module)
APIC-EM Path Trace application, 276–278
APIC-EM Plug and Play (PnP)
Configuration Upload page, 275
dashboard, 271–272
Device History, 273
overview, 269–278
Pending status information page, 274
project details, 272–273
Software Images tab, 274–275
Unplanned Devices screen, 275–276
authentication via Python
Host API, 299–301
Ticket API, 297–299
Device Inventory application, 281–282
Dynamic QoS (Quality of Service), 285–286
Easy QoS (Quality of Service), 283–285
IWAN (Intelligent WAN), 264–269
network devices, listing, 292–293
overview, 70, 263
Policy application, 286–287
Postman APIs
available APIs, 296
DevNet APIC-EM sandbox, 288
Host API, 291–292
Network Device API, 292–293
Application Spine Engine (ASE), 67

Ticket API, 288–291
User API, 294–296
Topology application, 279–281
users, viewing, 294–296
APIC-EM-AUTH.py script, 297–299
APIC-EM-SHOW-HOST.py script, 299–301
APIs (application programming interfaces)
APIC RESTful API. See also Cobra SDK
API inspector, 179–182
APIC automation with UCS Director, 211
APIC configuration automation, 188–191
atomic mode, 174–175
authentication, 185–188
components, 175–176
event subscription, 196–198
forgiving mode, 174–175
GUI, 178
invoking, 176–178
object save-as, 178–179
programmability, 192–196
read operations, 176
Visore, 182–185
write operations, 177
CLI APIs, 115–116
ConfD, 261
event subscription, 196–198
Host API
Postman API, 291–292
Python script, 299–301
NX-API
authentication, 136–138
automation tools, 151
CLI mode, 129–130
DevOps tools, 151
event subscription, 143–146
IP address configuration, 130–131
message format, 126
NX-API REST, 131–136
NXTool Kit, 146–151
object modification via Postman, 138–140
object modification via Python, 140–143
overview, 125
sandbox, 127–129
security, 126
transport, 125–126
Visore, 134–136
Open Weather Map API, 40–43
Postman, 292–293, 294–296
Python, 37
Ticket API
Postman, 288–291
Python script, 297–299
applets (EEM), 246–251
Application Centric Infrastructure (ACI). See Cisco ACI (Application Centric Infrastructure)
application leaf engine (ALE), 67
application network profiles (ANPs), 72
application policy infrastructure controller enterprise module. See APIC-EM (application policy infrastructure controller enterprise module)
application programming interfaces. See APIs (application programming interfaces)
Application Spine Engine (ASE), 67
Application-Centric Infrastructure (ACI)

EPGs (end point groups), 72–73
overview, 70–72
policy instantiation, 73–74

application-level health scores, 174

applications
installing
in Guestshell, 110
on Linux, 64–65

open-source applications, 64

Apt, 65

architecture
Cisco ACI (Application Centric Infrastructure), 169
Cisco data center, 76–80
Linux, 58
NDB (Nexus Data Broker), 74–76

arithmetic (Bash), 90–91

Arya (APIC REST Python Adapter)
AryaLogger, 207–208
installing, 207–208
WebArya, 211

AryaLogger
installing, 207–208
program example, 207–208

ASE (Application Spine Engine), 67

ASICs, 67–68

ASP (AutoSmart Ports)
availability, 216
device-specific macros and descriptions, 217
enabling on Cisco Catalyst Switch, 217–219
overview, 216–217
website, 220

asterisk (*), 32
atomic mode (ACI API), 174–175

audit trailing (ConfD), 262

authentication
APIC RESTful API
authentication methods, 185–186
overview, 182–186
with Python, 186–188

APIC-EM authentication
Host API, 291–292, 299–301
Ticket API, 288–291, 297–299

Cobra SDK, 201
NX-API, 136–138

auto qos global compact command, 231, 233
auto qos voip cisco-phone command, 233

Auto Security
availability, 230
enabling on Cisco Catalyst Switch, 228–230
overview, 227–228

AutoConf
availability, 220
enabling on Cisco Catalyst Switch, 222–224
interface templates and descriptions, 220–222
template modification, 224–227
website, 227

autocfg enable command, 222

automated management and monitoring
EEM (Embedded Event Manager)
EEM applets, 246–251
e-mail variables, 251
overview, 246
summary, 253
Tcl scripts, 251–252
website, 253
Smart Call Home
enabling on Cisco Catalyst Switch, 237–243
overview, 236–237
website, 239, 243
Tcl Shell, 243–246
automated port profiling
ASP (AutoSmart Ports)
device-specific macros and descriptions, 217
enabling on Cisco Catalyst Switch, 217–219
overview, 216–217
website, 220
Auto Security
availability, 230
enabling on Cisco Catalyst Switch, 228–230
overview, 227–228
AutoConf
availability, 220
enabling on Cisco Catalyst Switch, 222–224
interface templates and descriptions, 220–222
template modification, 224–227
website, 227
AutoSmart Ports, 216
overview, 216
automation tools. See also Cisco
ACI (Application Centric Infrastructure); off-box programmability (NX-OS); on-box programmability (NX-OS)
Ansible
Nexus 9000, 157–158
overview, 156–157
APIC-EM (application policy infrastructure controller enterprise module)
APIC-EM Plug and Play (PnP), 269–278
authentication via Python, 297–301
Device Inventory application, 281–282
Dynamic QoS (Quality of Service), 285–286
Easy QoS (Quality of Service), 283–285
IWAN (Intelligent WAN), 264–269
network devices, listing, 292–293
overview, 263
Path Trace application, 276–278
Policy application, 286–287
Postman APIs, 288–296
Topology application, 279–281
users, viewing, 292–293
ASP (AutoSmart Ports)
device-specific macros and descriptions, 217
enabling on Cisco Catalyst Switch, 217–219
overview, 216–217
website, 220
Auto Security
availability, 230
enabling on Cisco Catalyst Switch, 228–230
overview, 227–228
AutoConf
availability, 220
enabling on Cisco Catalyst Switch, 222–224
interface templates and descriptions, 220–222
template modification, 224–227
website, 227
AutoSmart Ports, 216
overview, 216
automation tools. See also Cisco
ACI (Application Centric Infrastructure); off-box programmability (NX-OS); on-box programmability (NX-OS)
interface templates and descriptions, 220–222
template modification, 224–227
website, 227
automation example, 11
AutoQoS on LAN devices
   enabling on Cisco Catalyst Switch, 231–233
   overview, 230–231
AutoQoS on WAN devices
   enabling on Cisco ISR routers, 234–236
   overview, 233–234
AutoSmart Ports, 216
ConfD, 259–263
DevOps, 304
EEM (Embedded Event Manager)
   EEM applets, 246–251
   email variables, 251
   overview, 246
   summary, 253
   Tcl scripts, 251–252
   website, 253
importance of, 303
NETCONF (network configuration protocol), 258–259
NX-OS support for, 151
overview, 255–256
Puppet
   ciscopuppet, 154–156
   Nexus 9000, 154–157
   overview, 152–153
Smart Call Home
   enabling on Cisco Catalyst Switch, 237–243
   overview, 236–237
   website, 239, 243
Tcl Shell, 243–246
when to use, 10
YANG (yet another next generation) data models, 256–258
automation versus orchestration, 6
AutoSmart Ports. See ASP (AutoSmart Ports)
Awk, 98–99

B

backslash (\), 90
bandwidth allocation, 285
Bash scripts
   arithmetic, 90–91
   Awk, 98–99
   commands
      boot_config.sh, 106
      chkconfig, 105–106
      man, 99–100
   conditions, 91–94
   ethtool, 103
   flow control, 91–94
   ifconfig, 101
   loops, 93–94
   Nexus 9000, 99–100
   NTP server configuration, 106
   operators, 91–92
   overview, 56, 88–89
   redirection, 94–96
   running at startup, 103–106
   Sed text editor, 96–98
   tcpdump, 101–103
   variables, 89–90
BGP configuration with NXTool Kit, 148–151
/bin directory, 64
boot, configuring NTP servers at, 106
/boot directory, 64
boot order pxe bootflash
command, 88
boot_config.sh command, 106
/bootflash directory, 118
BPDUs (bridge protocol data units), 78
bridge protocol data units (BPDUs), 78
Broadcom, NFE (network forwarding engine), 67
browsers, Visore, 134–136
BUM traffic, 80

C

call-home command, 239–241
call-home send alert-group inventory profile CiscoTAC-1 command, 242–243
calling functions, 29
calls (Postman), creating, 189–191
campus environments, automation tools for
APIC-EM (application policy infrastructure controller enterprise module)
   APIC-EM Plug and Play (PnP), 269–278
   authentication via Python, 297–301
   Device Inventory application, 281–282
   Dynamic QoS (Quality of Service), 285–286
   Easy QoS (Quality of Service), 283–285
   IWAN (Intelligent WAN), 264–269
   network devices, listing, 292–293
   overview, 263
   Path Trace application, 276–278
   Policy application, 286–287
   Postman APIs, 288–296
   Topology application, 279–281
   users, viewing, 292–293
ConfD, 259–263
NETCONF (network configuration protocol), 258–259
overview, 255–256
YANG (yet another next generation) data models, 256–258
capturing IP packet headers, 101–103
caret (\^), 32
case sensitivity (Cisco ACI API), 177
case statement, 92–93
cat command, 57
catalogs, Cisco Prime Service Catalog, 213
Catalyst Switch. See Cisco Catalyst Switch, enabling automation tools on
cd command, 57
CDB Database (ConfD), 261
cdp_neighbor.py, 121–123
changing dictionary data, 28
Chef, 151
chkconfig, 105–106
chmod command, 29, 57, 64
chown command, 57
chvrf command, 109
CI/CD (continuous integration/continuous deployment), 2
ASP (AutoSmart Ports), 217–219
Auto Security, 228–230
AutoConf, 222–224
AutoQoS on LAN devices, 231–233
Smart Call Home, 237–243
Cisco Data Center (GitHub), 123, 207
Cisco data center networking
ACI (Application Centric Infrastructure). See Cisco ACI (Application Centric Infrastructure)
Cisco Nexus Fabric Manager, 80–81
Cisco VTS (Virtual Topology System), 81
network architecture, 76–80
Nexus 9000, 67–70
Nexus Data Broker (NDB) architecture, 74–76
RESTful API, 75
use case, 75–76
NX-OS. See Cisco Nexus NX-OS
Cisco DevNet
APIC-EM sandbox, 288
website, 204
Cisco Ignite, 87–88
Cisco ISR routers, enabling AutoQoS on, 234–236
Cisco IWAN (Intelligent WAN), 264–269
Cisco Merchant + strategy, 67
Cisco Netflow Generation Appliance, 75
Cisco Nexus Fabric Manager, 80–81
Cisco Nexus NX-OS
Ansible
Nexus 9000, 157–158
overview, 156–157
Bash scripts
arithmetic, 90–91
Awk, 98–99
conditions, 91–94
ethtool, 103
flow control, 91–94
ifconfig, 101
loops, 93–94
Nexus 9000, 99–100
NTP server configuration, 106
operators, 91–92
overview, 88–89
redirection, 94–96
running at startup, 103–106
Sed text editor, 96–98
tcpdump, 101–103
variables, 89–90
Cisco Ignite, 87–88
Cisco libraries, importing, 113–115
EEM (Embedded Event Manager) actions, 112–113
leveraging Python scripts from, 121–123
neighbor discovery, 121–123
system events, 112
variables, 113
Guestshell
application installation, 110
network access, 109–110
NMap installation, 111
overview, 108–109
Puppet agent installation, 111
iPXE, 88
LXC (LinuX Containers), 106–107
NDB (Nexus Data Broker), 111–112
NX-API authentication, 136–138
automation tools, 151
CLI mode, 129–130
DevOps tools, 151
event subscription, 143–146
IP address configuration, 129–130
message format, 126
NX-API REST, 131–136
NXTool Kit, 146–151
object modification via Postman, 138–140
object modification via Python, 140–143
overview, 125
sandbox, 127–129
security, 126
transport, 125–126
Visore, 134–136
overview, 69, 83
POAP (power-on auto provisioning), 83–87
Puppet
Nexus 9000, 154–157
overview, 152–153
Python scripts
Cisco Python package, 116–117
CLI APIs, 115–116
leveraging from EEM syslog event, 121–123
non-interactive Python, 118
Reload_in Pseudocode, 118–121
Cisco POAP (power-on auto provisioning), 83–87
Cisco Prime Service Catalog, 213
Cisco Python package, 116–117
Cisco Smart Call Home
enabling on Cisco Catalyst Switch, 237–243
overview, 236–237
website, 239, 243
Cisco Smart Install (SMI) Proxy, 271
Cisco Switched Port Analyzer (SPAN), 74
Cisco UCS (Unified Computing System) Manager, 7
Cisco Validated Designs (CVDs), 266
Cisco VTS (Virtual Topology System), 81
ciscopuppet
documentation, 156
installing, 154
NX-OS configurations, 155–156
verifying, 154
cisco.vrf module, 116–117
clear command, 57
cleared faults, 174
cli(), 115
CLI APIs, 115–116
CLI mode (NX-API), 129–130
$_cli_result keyword, 247
clid(), 115
clip(), 115
Cliqr, 7
cloning Git repositories, 44–47
cloud operational models
network programmability, 304
overview, 6–8
cmd dictionary value, 129
Cobra SDK
Arya (APIC REST Python Adapter)
installing, 207–211
WebArya, 211
authentication, 201
debug event manager all command 317

documentation, 198
installing, 199–200
libraries, 200–201
modules, 200
objects, 202–204
tenant configuration example, 204–207
cobra.mit library, 201
cobra.model library, 201
collections, 188
comments (Python), 29
committing files to GitHub, 48
concatenating strings, 23

conditions
   Bash scripting, 91–94
   explained, 17
   Python conditions, 24–25
conf command, 130
ConfD, 259–263
config false command, 258
ConfigRequest(), 202
configuration
   ACI tenant configuration with Cobra, 204–207
   APIC configuration automation, 188–191
   BGP configuration with NXTool Kit, 148–151
   IP addresses, 130–131
   NTP server, 106
   POAP (power-on auto provisioning), 83–87
   Puppet, 155
Configuration Upload page
   (APIC-EM PnP), 275
configure terminal command, 218, 247
consumption model (DevOps), 304
continuous integration/continuous deployment (CI/CD), 2
converting variables to strings, 23
cookies, nxapi_auth, 126, 136–138
Core Engine (ConfD), 261
cp command, 57
critical faults, 173
custom tags, 170
customized virtual environments, 163
CVDs (Cisco Validated Designs), 266

D

DAI (Dynamic ARP inspection), 227–228
dashboard (APIC-EM PnP), 271–272
data center networking. See Nexus
data center networking
data management engine (DME), 133, 174
data models, YANG (yet another next generation), 256–258
Data Provider API, 261
data types, 15–16, 27–28
data-encoding formats
   JSON (Javascript Object Notation), 39–45
   XML (Extensible Markup Language), 38–39
datetime library, 30, 35–36
deactivate command, 163–164
debug event manager action cli command, 239–241, 247
debug event manager action mail command, 249
debug event manager all command, 249
debugging
- HTML APIC GUI, 178
- PyCharm, 54–55

depth packet inspection (DPI) tool, 4

def keyword, 29

defining
  - functions, 29
  - variables, 15–16

DELETE method, 37

development. See software development

Device History (APIC-EM PnP), 273

Device Inventory application (APIC-EM), 281–282

DevNet
  - APIC-EM sandbox, 288
  - website, 66

DevOps
  - network consumption model, 304
  - NX-OS support for, 151

df command, 57

DHCP (Dynamic Host Configuration Protocol)
  - DHCP Snooping, 227
  - POAP configuration, 84–85

dictionaries, 28

dir command, 147–148

directories
  - /bin, 64
  - /boot, 64
  - /bootflash, 118
  - /etc, 64
  - /etc/passwd, 57
  - /home, 64
  - overview, 64, 118
  - /sbin, 64
  - /usr, 64
  - /var/log, 64

discovery
  - EEM (Embedded Event Manager), 121–123
  - MOs (managed objects), 178

displaying
  - APIC-EM users, 294–296
  - Linux processes, 59–61

distinguished names (DNs), 133, 170

DME (data management engine), 133, 174

DMP _INTERFACE _TEMPLATE, 220

DNs (distinguished names), 133, 170

documentation
  - ciscopuppet, 156
  - Cobra SDK, 198
  - man documentation, 99
  - NFM (Nexus Fabric Manager), 82
  - Python, 24
  - VTS (Virtual Topology System), 82

dohost command, 109

dollar sign ($), 32

downloading Git, 45

DPI (deep packet inspection) tool, 4

Dynamic ARP inspection (DAI), 227–228

Dynamic Host Configuration Protocol. See DHCP (Dynamic Host Configuration Protocol)

Dynamic QoS (Quality of Service), 285–286

E

Easy QoS (Quality of Service), 283–285

ECMP (Equal Cost Multiple Path), 77
editing
JSON editor, 178
source code, 49–50
text in Bash
  *Awk*, 98–99
  *Sed* text editor, 96–98
editors
JSON editor, 178, 191
Sed, 96–98
EEM (Embedded Event Manager)
actions, 112–113
EEM applets, 246–251
email variables, 251
leveraging Python scripts from,
  121–123
neighbor discovery, 121–123
overview, 246
summary, 253
system events, 112
Tcl scripts, 251–252
variables, 113
website, 253
.egg files, 198
else statements
overview, 17
Python, 24–25
e-mail variables (EEM), 251
Embedded Event Manager. *See* EEM
  (Embedded Event Manager)
enable command
  Auto Security, 228
  AutoConf, 224
  AutoSmart Ports, 217
  EEM applets, 247
enabling
  ASP (AutoSmart Ports), 217–219
  Auto Security, 228–230
  AutoConf, 222–224
  AutoQoS on LAN devices, 231–233
  AutoQoS on WAN devices, 234–236
  Smart Call Home, 237–243
end point groups (EPGs)
  assigning interfaces to, 193–196
  definition of, 72–74, 161–162
environments. *See* campus
  environments, automation tools
  for; virtual environments
EPGs (end point groups)
  assigning interfaces to, 193–196
  definition of, 72–74, 161–162
eq operator, 91
Equal Cost Multiple Path (ECMP), 77
equal sign (=), 92
errors. *See* exception handling
/etc directory, 64
/etc/passwd file, 57
Ethernet VPNs (EVPNs), 80–81
ethtool, 103
event manager run command,
  251–252
event manager session cli username
  command, 247
event none command, 251–252
events
  Cisco ACI event subscription,
    196–198
  EEM (Embedded Event Manager), 112
  NX-API event subscription,
    143–146
eVPN, 80
EVPNs (Ethernet VPNs), 80–81
exception handling
  definition of, 162
  examples, 166–168
virtual environments, 163–165
virtualenv installation, 162–163
exiting virtual environments, 163–164
expressions, regular
  Python, 31–37
  Sed support for, 96
Extensible Markup Language (XML), 38–39

F

Fabric Manager, 80–81
faults
  lifecycle, 171–173
  severity, 173–174
  states, 172–173
fields (Path Trace), 277
FIFO (first-in, first-out), 176
file prompt quiet command, 250
file system, 63–64
files
  .egg files, 198
  adding to GitHub, 47–48
  .py files, 29
  Python files, 29
filters, tcpdump, 102
find method, 22
first-in, first-out (FIFO), 176
flow control
  Bash scripting, 91–94
  conditions
  explained, 17
  Python conditions, 24–25
  loops
  explained, 18
  Python loops, 25–28

forgiving mode (ACI API), 174–175
formatted data. See data-encoding formats
formatting tools (JSON), 178, 191
freezing virtual environments, 164–165
functions
  calling, 29
  defining, 29
  overview, 18–19, 28–29

G
gateways, transport, 236–237
General Public License (GPL), 56
GET method, 37
Git
  commands, 49
  downloading, 45
  files, adding to repositories, 47–48
  overview, 45
  resources, 49
  git add command, 47–48, 49
  git clone command, 44–47, 49
  git commit command, 48, 49
  git pull command, 49
  git push command, 48, 49
  git status command, 47
GitHub
  Cisco Data Center page, 123, 207
  overview, 45
  repository creation, 45–47
  repository updates, 47–48
  resources, 49
Google Postman, 40–43
GPL (General Public License), 56
grep command, 57
gt operator, 91
Guestshell
  application installation, 110
  network access, 109–110
  NMap installation, 111
  overview, 108–109
  Puppet agent installation, 111
GUI (APIC), 178

handling exceptions. See exception handling
hashes, 17
health scores, 174
hello_world (Bash), 104–105
help
  help command, 147–148
  help method, 23
  Linux, 65–66
    Python, 23–24, 116–117
help command, 147–148
help method, 23
/home directory, 64
$HOME variable, 89
Host API
  Postman API, 291–292
    Python script, 299–301
human language versus machine language, 8–9
IaaS (infrastructure as a service), 6–7
if statements
  Bash scripting, 91–92
  if-else, 17, 24–25
  if-then, 246–247
  overview, 17
    Python, 24–25
ifconfig, 101
if-else statements, 17, 24–25
if-then statements, 246–247
Ignite, 87–88
import command, 30
imports
  Cisco libraries, 113–115
  Cobra SDK libraries, 200–201
  NXTool Kit, 146
  Python libraries, 30
info faults, 174
information technology as a service (ITaaS), 213
infrastructure as a service (IaaS), 6–7
innovation (network), 4–6
installation
  applications
    in Guestshell, 110
    on Linux, 64–65
    Arya (APIC REST Python Adapter), 207–208
    AryaLogger, 207–208
ciscopuppet, 154
Cobra SDK, 200
libraries into virtual environments, 164
NDB (Nexus Data Broker), 111–112
NMap in Guestshell, 111
NXTool Kit, 146
Puppet agents in Guestshell, 111
Python, 20
Python libraries, 30–31
virtualenv, 162–163
instantiation

ACI (Application Centric Infrastructure) policy, 73–74, 161–162
objects, 20
Intelligent WAN (IWAN), 264–269
interface command, 130
interface status, checking, 101
interfaces
assigning to EPGs (end point groups), 193–196
AutoConf interface templates, 220–222
invoking APIC RESTful API, 176–178
IP address configuration, 130–131
IP packet headers, capturing, 101–103
IP phone port profiling. See port profiling
IP_CAMERA_INTERFACE_TEMPLATE, 220
IP_PHONE_INTERFACE_TEMPLATE, 220
iPXE, 88
ISR routers, enabling AutoQoS on, 234–236
ITaaS (information technology as a service), 213
IWAN (Intelligent WAN), 264–269

K

keywords. See also statements
$_cli_result, 247
def, 29
kill command, 57

L

L2-based network architecture, 77–79
LANs, AutoQoS on LAN devices
enabling on Cisco Catalyst Switch, 231–233
overview, 230–231
LAP_INTERFACE_TEMPLATE, 220
layer 2-based network architecture, 77–79
Lean IT, 303
less command, 57
libraries
Cisco libraries, importing, 113–115
Cobra SDK, 200–201
installing into virtual environments, 164
Python libraries
datetime, 35–36
importing, 30
installing, 30–31
json, 44–45
requests, 43–44
sys, 34–35
lifecycle of faults, 171–173
Linux
architecture, 58
bash, 56
commands

J

Javascript Object Notation. See JSON (Javascript Object Notation)
JSON (Javascript Object Notation)
formatting tools, 178, 191
JSON-RPC, 126
overview, 39–45
json library, 30, 44–45
jsoneditoronline.org, 139
management information model (Cisco ACI) 323

chmod, 64
command summary, 56–58
ls, 63
systemd, 61–62
top, 61
directories, 64
file system, 63–64
GPL (General Public License), 56
help, 65–66
installing applications on, 64–65
LXC (LinuX Containers), 62
overview, 55–56
permissions, 63–64
processes, displaying, 59–61
ps tool, 59
resources, 65–66
systemd, 61–62
yum tool, 65
LinuX Containers (LXC), 62, 106–107. See also Guestshell
list command, 107
lists
list command, 107
overview, 16
in Python, 27
lookupByClass method, 202–203
lookupByDN method, 202–204
for loops
Bash scripting, 93
overview, 18
in Python, 25–26
loops
Bash scripting, 93–94
explained, 18, 93–94
looping through dictionaries, 28
for loops
Bash scripting, 94
overview, 18
Python loops, 26–27
ls command, 57, 63
lt operator, 91
LXC (LinuX Containers), 62, 106–107. See also Guestshell
M
M2M (machine-to-machine interactions), 9–10
machine language versus human language, 8–9
machine-to-machine interactions (M2M), 9–10
MACs (moves, additions, and changes), 216
major faults, 173
man command, 57, 99
man documentation, 99
Managed Object API, 261
Managed Object module (Cobra SDK), 200
managed objects (MOs)
Cobra SDK Managed Object module, 200
definition of, 133, 169–170
discovering, 178
health scores, 174
Managed Object API, 261
management information model (Cisco ACI)
fault lifecycle, 171–173
fault severity, 173–174
health scores, 174
MIT (management information tree), 169–170
MOs (managed objects), 169–170
object names, 170
management information tree (MIT), 169–170
management tasks
EEM (Embedded Event Manager)
   EEM applets, 246–251
   email variables, 251
   overview, 246
   summary, 253
   Tcl scripts, 251–252
   website, 253
Smart Call Home
   enabling on Cisco Catalyst Switch, 237–243
   overview, 236–237, 243
   website, 239
Tcl Shell, 243–246
MD5 hash, 86
Merchant + strategy (Cisco), 67
message format (NX-API), 125–126
methods
   aaaLogin, 185
   aaaLogout, 186
   aaaRefresh, 185
   APIC authentication methods, 185–186
   ConfigRequest(), 202
   find, 22
   help, 23
   lookupByClass, 202–203
   lookupByDN, 202–204
   re.findall, 32
   re.match, 33
   re.search, 33–34
   re.sub, 34
   split, 22
minor faults, 173
MIT (management information tree), 169–170
mkdir command, 57
models, YANG (yet another next generation), 256–258
modification
   NX-API objects
      via Postman, 138–140
      via Python, 140–143
   modifying AutoConf templates, 224–227
modules (Cobra SDK), 200
monitoring tasks
EEM (Embedded Event Manager)
   EEM applets, 246–251
   email variables, 251
   overview, 246
   summary, 253
   Tcl scripts, 251–252
   website, 253
Smart Call Home
   enabling on Cisco Catalyst Switch, 237–243
   overview, 236–237
   website, 239, 243
Tcl Shell, 243–246
more command, 252
MOs (managed objects)
   Cobra SDK Managed Object module, 200
definition of, 133, 169–170
discovering, 178
health scores, 174
Managed Object API, 261
moves, additions, and changes (MACs), 216
MSP_CAMERA_INTERFACE_TEMPLATE, 220
MSP_VC_INTERFACE_TEMPLATE, 220
mutability of data in Python, 27
mv command, 57

N

names
  object names, 170
  variables, 15–16
Naming module (Cobra SDK), 200
NDB (Nexus Data Broker)
  architecture, 74–76
  overview, 111–112
  RESTful API, 75
  use case, 75–76
ne operator, 91
neighbor discovery (EEM), 121–123
NETCONF (network configuration protocol), 258–259
Netflow Generation Appliance, 75
network automation tools
  APIC-EM (application policy infrastructure controller enterprise module)
  APIC-EM Plug and Play (PnP), 269–278
  authentication via Python, 297–301
  Device Inventory application, 281–282
  Dynamic QoS (Quality of Service), 285–286
  Easy QoS (Quality of Service), 283–285
  IWAN (Intelligent WAN), 264–269
  network devices, listing, 292–293
  overview, 263
  Path Trace application, 276–278
  Policy application, 286–287
  Postman APIs, 288–296
  Topology application, 279–281
  users, viewing, 292–293
automation example, 11
ConfD, 259–263
DevOps, 304
importance of, 303
NETCONF (network configuration protocol), 258–259
overview, 255–256
when to use, 10
YANG (yet another next generation) data models, 256–258
network configuration protocol (NETCONF), 258–259
network consumption model (DevOps), 304
network controllers. See also Cisco ACI (Application Centric Infrastructure)
  Cisco Nexus Fabric Manager, 80–81
  Cisco VTS (Virtual Topology System), 81
Network Device API, 292–293
network devices (APIC-EM), listing, 292–293
network engineers, skill set needed by, 303–304
network forwarding engine (NFE), 67
network innovation, 4–6
network programmability
automation tools. See network automation tools
benefits of
  network innovation, 4–6
  simplified networking, 4
Cisco ACI (Application Centric Infrastructure). See Cisco ACI (Application Centric Infrastructure)
cloud operational models, 6–8, 304
definition of, 3
importance of, 303–305
off-box programmability (NX-OS). See off-box programmability (NX-OS)
on-box automation tools. See on-box programmability (NX-OS)
origin of, 9–10
overview, 1–2
SDN (software-defined networking), 8–9
network programmability user group (NPUG), 66
Nexus 9000
  Ansible, 157–158
  Bash scripting, 99–100
  network architecture, 76–80
  overview, 67–70
  Puppet, 154–157
Nexus 9200K, 68
Nexus 9300EX, 68
Nexus Data Broker (NDB)
  architecture, 74–76
  overview, 111–112
  RESTful API, 75
  use case, 75–76
Nexus data center networking.
  See also Cisco ACI (Application Centric Infrastructure)
Cisco Nexus Fabric Manager, 80–81
Cisco VTS (Virtual Topology System), 81
network architecture, 76–80
Nexus 9000, 67–70
Nexus Data Broker (NDB)
  architecture, 74–76
  RESTful API, 75
  use case, 75–76
NX-OS. See Cisco Nexus NX-OS
Nexus Fabric Manager, 80–81
Nexus NX-OS. See Cisco Nexus NX-OS
NFE (network forwarding engine), 67
NFM (Nexus Fabric Manager), 80–81
NMap agent installation, 111
no shutdown command, 247
non-interactive Python, 118
NPUG (network programmability user group), 66
NTP server configuration, 106
NX-API
  Ansible, 156–157
  authentication, 136–138
  automation tools, 151
  CLI mode, 129–130
  DevOps tools, 151
  event subscription, 143–146
  IP address configuration, 129–130
  message format, 126
NX-API REST
  CLI versus object model,
    131–132
  logical hierarchy, 132–133
  overview, 131–136
NXTool Kit

BGP configuration, 148–151
importing, 146
installing, 146
usage example, 146–148

object modification via Postman, 138–140
object modification via Python, 140–143
overview, 69, 125
sandbox, 127–129
security, 126
transport, 125–126
Visore, 134–136

NX-API REST

CLI versus object model, 131–132
logical hierarchy, 132–133
overview, 131–136

nxapi_auth cookie, 126, 136–138

NX-OSS, See Cisco Nexus NX-OS

NXTool Kit

BGP configuration, 148–151
importing, 146
installing, 146
usage example, 146–148

Object Identifiers (OIDs), 258

object modification (NX-API)

via Postman, 138–140
via Python, 140–143

object save-as (APIC), 178–179
object store browser (APIC), 182–185

object-oriented data models

CLI versus object model, 131–132

logical hierarchy, 132–133
overview, 131–136

objects

Cobra SDK, 202–204
instantiating, 20
MOs (managed objects)

definition of, 133, 169–170
discovering, 178
health scores, 174
names, 170
overview, 19–20

off-box programmability (NX-OSS)

Ansible

Nexus 9000, 157–158
overview, 156–157
definition of, 125

NX-API

authentication, 136–138
automation tools, 151
CLI mode, 129–130
DevOps tools, 151
event subscription, 143–146
IP address configuration, 129–130
message format, 126

NX-API REST, 131–136

NXTool Kit, 146–151

object modification via Postman, 138–140
object modification via Python, 140–143
overview, 125
sandbox, 127–129
security, 126
transport, 125–126
Visore, 134–136

Puppet, 152–157
OIDs (Object Identifiers), 258

on-box automation tools

ASP (AutoSmart Ports)
- device-specific macros and descriptions, 217
- enabling on Cisco Catalyst Switch, 217–219
- overview, 216–217
- website, 220

Auto Security
- availability, 230
- enabling on Cisco Catalyst Switch, 228–230
- overview, 227–228

AutoConf
- availability, 220
- enabling on Cisco Catalyst Switch, 222–224
- interface templates and descriptions, 220–222
- template modification, 224–227
- website, 227

AutoQoS on LAN devices
- enabling on Cisco Catalyst Switch, 231–233
- overview, 230–231

AutoQoS on WAN devices
- enabling on Cisco ISR routers, 234–236
- overview, 233–234

AutoSmart Ports, 216

EEM (Embedded Event Manager)
- EEM applets, 246–251
- email variables, 251
- overview, 246
- summary, 253
- Tcl scripts, 251–252
- website, 253
- overview, 215

Smart Call Home
- enabling on Cisco Catalyst Switch, 237–243
- overview, 236–237
- website, 239, 243

Tcl Shell, 243–246

on-box programmability (NX-OS)

Bash scripts
- arithmetic, 90–91
- Awk, 98–99
- conditions, 91–94
- ethtool, 103
- flow control, 91–94
- ifconfig, 101
- loops, 93–94
- Nexus 9000, 99–100
- NTP server configuration, 106
- operators, 91–92
- overview, 88–89
- redirection, 94–96
- running at startup, 103–106
- Sed text editor, 96–98
- tcpdump, 101–103
- variables, 89–90

Cisco Ignite, 87–88

EEM (Embedded Event Manager)
- actions, 112–113
- leveraging Python scripts from, 121–123
- neighbor discovery, 121–123
- system events, 112
- variables, 113

Guestshell
- application installation, 110
- network access, 109–110
- NMap installation, 111
Plug and Play (PnP) 329

overview, 108–109
Puppet agent installation, 111
iPXE, 88
LXC (LinuX Containers), 106–107
NDB (Nexus Data Broker), 111–112
overview, 83
POAP (power-on auto provisioning), 83–87
Python scripts
Cisco libraries, importing, 113–115
Cisco Python package, 116–117
CLI APIs, 115–116
leveraging from EEM syslog event, 121–123
non-interactive Python, 118
Reload_in Pseudocode, 118–121
online resources
ciscopuppet, 156
Git, 49
Linux, 65–66
Python, 36–37
Open NX-OS, 6
Open Weather Map API example, 40–43
open-source applications, 64
operators
Bash, 91–92
Python, 25
orchestration, 6
os library, 30

package management tool. See PIP (package management tool)
packets
IP packet headers, capturing, 101–103
VXLAN packets, 79–80
Path Trace application (APIC-EM), 276–278
path traces, 276–278
$PATH variable, 89
Pending status information page (APIC-EM PnP), 274
per-hop behaviors (PHBs), 285
period (.), 32, 33
permissions (Linux), 63–64
PHBs (per-hop behaviors), 285
pings, aborting, 245
PPIP (package management tool)
overview, 30–31
virtual environments
freezing, 164–165
installing, 164
recreating, 165
sharing, 165
pip freeze command, 165
pip install command, 31, 165, 208
pip list command, 165
pip search command, 31
pip show command, 31
platform as a service (PaaS), 6–7
Plug and Play (PnP)
APIC-EM Plug and Play (PnP)
Configuration Upload page, 275
dashboard, 271–272
overview, 269–271, 269–278

PaaS (platform as a service), 6–7
Pending status information page, 274
project details, 272–273
Software Images tab, 274–275
Unplanned Devices screen, 275–276
Device History, 273

plus sign (+), 32

PnP (Plug and Play)

APIC-EM Plug and Play (PnP)
Configuration Upload page, 275
dashboard, 271–272
Device History, 273
overview, 269–271, 269–278
Pending status information page, 274
project details, 272–273
Software Images tab, 274–275
Unplanned Devices screen, 275–276

POAP (power-on auto provisioning), 83–87
Policy application (APIC-EM), 286–287

policy instantiation
ACI (Application Centric Infrastructure), 73–74
Cisco ACI (Application Centric Infrastructure), 161–162

port profiling
ASP (AutoSmart Ports)
device-specific macros and descriptions, 217
enabling on Cisco Catalyst Switch, 217–219
overview, 216–217

Auto Security
availability, 230

enabling on Cisco Catalyst Switch, 228–230
overview, 227–228

AutoConf
enabling on Cisco Catalyst Switch, 222–224
interface templates and descriptions, 220–222
template modification, 224–227
overview, 216

Port Security, 228

POST method, 37

Postman
APIC configuration automation, 188–191

APIC-EM APIs
available APIs, 296
DevNet APIC-EM sandbox, 288
Host API, 291–292
Network Device API, 292–293
Ticker API, 288–291
User API, 294–296
calls, creating, 189–191
collections, 188
NX-API object modification, 138–140
Open Weather Map API example, 40–43
pound sign (#), 29

power-on auto provisioning. See
POAP (power-on auto provisioning)

Prime Service Catalog, 213
print command, 21, 128–129
PRINTER_INTERFACE TEMPLATE, 220
processes (Linux), displaying, 59–61
profiling ports. See port profiling
programmability. See network programmability; off-box programmability (NX-OS); on-box programmability (NX-OS)

project details (APIC-EM PnP), 272–273

provisioning. See POAP (power-on auto provisioning)

ps tool, 59

pseudocode, 14

Puppet
agent installation in Guestshell, 111
ciscopuppet
  documentation, 156
  installing, 154
  NX-OS configurations, 155–156
  verifying, 154
Nexus 9000, 154–157
  overview, 152–153
PUT method, 37
pwd command, 57
$PWD variable, 89
.py filename extension, 29

PyCharm
  debugging, 54–55
  interface, 50–53
  virtualenv, 166
  writing code in, 53

Python
  APIC authentication, 186–188
  APIC-EM authentication
    Host API, 299–301
    Ticket API, 297–299
  APIs (application programming interfaces), 37
  commands
    chmod, 29
    dir, 147–148
    help, 147–148
    import, 30
    pip install, 31
    pip search, 31
    pip show, 31
    print, 21
    str, 23
  comments, 29
  conditions, 24–25
  dictionaries, 28
  documentation, 24
  exception handling
    definition of, 162
    examples, 166–168
    virtualenv, 162–166
  files, 29
  functions, 28–29
  help, 23–24, 116–117
  installing, 20
  libraries
    Cisco libraries, importing, 113–115
    Cisco Python package, 116–117
    CLI APIs, 115–116
    datetime, 35–36
    importing, 30
    installing, 30–31
    json, 44–45
    requests, 43–44
    sys, 34–35
  lists, 27
  loops
    for loop, 25–26
    while loop, 26–27
methods
  - find, 22
  - help, 23
  - re.findall, 32
  - re.match, 33
  - re.search, 33–34
  - re.sub, 34
  - split, 22
non-interactive Python, 118

NX-API in
  - authentication, 136–138
  - CLI mode, 129–130
  - event subscription, 143–146
  - IP address configuration, 130–131
  - object modification, 140–143

online resources, 36–37
operators, 25
overview, 20–21
PIP (package management tool), 30–31

PyCharm
  - debugging, 54–55
  - interface, 50–53
  - writing code in, 53

regular expressions, 31–37
scripts
  - cdp_neighbor.py, 121–123
  - leveraging from EEM syslog event, 121–123
  - Reload_in Pseudocode, 118–121
  - scheduling, 57
  - SimpleMat.py example, 36
  - WebSoc.py, 144–145

SDKs (software development kits), 37
tuples, 27

variables
  - converting to strings, 23
  - overview, 20–21
  - strings, 22–23

web technologies
  - Google Postman, 40–43
  - JSON (Javascript Object Notation), 39–45
  - REST (Representational State Transfer), 37–38
  - XML (Extensible Markup Language), 38–39

python command, 20

Q

QoS (Quality of Service)
  - AutoQoS on LAN devices
    - enabling on Cisco Catalyst Switch, 231–233
    - overview, 230–231
  - AutoQoS on WAN devices
    - enabling on Cisco ISR routers, 234–236
    - overview, 233–234
  - on box-by-box basis, 259–260
  - Dynamic QoS (Quality of Service), 285–286
  - Easy QoS (Quality of Service), 283–285
  - network programmability and, 4
  - website, 230

Quality of Service. See QoS (Quality of Service)

R

raised state (faults), 172
raised-clearing state (faults), 173
$RANDOM variable, 89
re library, 30
read operations (Cisco ACI), 176
recreating virtual environments, 165
Redhat Package Manager (RPM), 64
redirection (Bash), 94–96
re.findall method, 32
registering your book, A00.0102
regular expressions
  Python, 31–37
    Sed support for, 96
relative names (RNs), 170
reload in command, 118–121
Reload in Pseudocode, 118–121
re.match method, 33
Remote Procedure Call (RPC),
  JSON-RPC, 126
repositories (GitHub)
  cloning, 44–47
  creating, 45–47
  updating, 47–48
Representational State Transfer.
  See REST (Representational State Transfer)
Request module (Cobra SDK), 200
requests library, 30, 43–44
re.search method, 33–34
resources
  ciscopuppet, 156
  Git, 49
  Linux, 65–66
  Python, 36–37
  software development, 65–66
REST (Representational State Transfer).
  See also APIC RESTful API; NX-API
Google Postman, 40–43
JSON (Javascript Object Notation),
  overview, 37–38
XML (Extensible Markup Language),
  38–39
re.sub method, 34
retaining state (faults), 173
return on investment (ROI) for software development, 13–14
reverse path traces, 278
rm command, 57
RNs (relative names), 170
ROI (return on investment) for software development, 13–14
roles (ConfD), 262
rollback management (ConfD), 262
ROUTER_INTERFACE TEMPLATE, 220
routers, enabling AutoQoS on, 234–236
RPC (Remote Procedure Call),
  JSON-RPC, 126
RPM (Redhat Package Manager), 64
run guestshell command, 108
running
  Bash scripts at startup, 103–106
  Python files, 29
SaaS (software as a service), 6–7
SaltStack, 151
sandbox (NX-API), 127–129
/sbin directory, 64
scheduling Python scripts, 57
scores (health), 174
scrapy library, 30
scripts

Bash

- arithmetic, 90–91
- Awk, 98–99
- conditions, 91–94
- ethtool, 103
- flow control, 91–94
- ifconfig, 101
- loops, 93–94
- Nexus 9000, 99–100
- NTP server configuration, 106
- operators, 91–92
- overview, 88–89
- redirection, 94–96
- running at startup, 103–106
- Sed text editor, 96–98
- tcpdump, 101–103
- variables, 89–90

Python

- APIC-EM-AUTH.py, 297–299
- APIC-EM-SHOW-HOST.py, 299–301
- cdp_neighbor.py, 121–123
- Cisco libraries, importing, 113–115
- Cisco Python package, 116–117
- CLI APIs, 115–116
- leveraging from EEM syslog event, 121–123
- non-interactive Python, 118
- Reload_in Pseudocode, 118–121
- scheduling, 57
- WebSoc.py, 144–145

SDKs (software development kits)

- Cobra SDK

Arya (APIC REST Python Adapter), 207–211
- authentication, 201
- documentation, 198
- installing, 200
- libraries, 200–201
- modules, 200
- objects, 202–204
- tenant configuration example, 204–207

Python, 37

SDN (software-defined networking), 8–9, 304

SD-WAN (software-defined WAN), 264–269

searching, strings, 22

secure unique device identification (SUDI), 270

security

- Auto Security
  - availability, 230
  - enabling on Cisco Catalyst Switch, 228–230
  - overview, 227–228
- NX-API, 126

Sed text editor, 96–98

Services module (Cobra SDK), 200

Session module (Cobra SDK), 200

set command, 89

severity of faults, 173–174

sharing virtual environments, 165

shells, Tcl Shell, 243–246

show auto qos command, 231–232, 234–235

show auto qos voip cisco-phone configuration command, 233

show auto security command, 228
<table>
<thead>
<tr>
<th>Command/Function/Feature</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>show auto security configuration command</td>
<td>229–230</td>
</tr>
<tr>
<td>show call-home command</td>
<td>242–243</td>
</tr>
<tr>
<td>show derived-config interface command</td>
<td>221, 223–225, 226</td>
</tr>
<tr>
<td>show interface command</td>
<td>128, 247</td>
</tr>
<tr>
<td>show log command</td>
<td>273</td>
</tr>
<tr>
<td>show macro auto device phone command</td>
<td>218</td>
</tr>
<tr>
<td>Show Reverse button (Path Trace application)</td>
<td>278</td>
</tr>
<tr>
<td>show running-config command</td>
<td>225–226, 273</td>
</tr>
<tr>
<td>show running-config interface command</td>
<td>223</td>
</tr>
<tr>
<td>show shell functions command</td>
<td>218</td>
</tr>
<tr>
<td>show shell triggers command</td>
<td>218–219</td>
</tr>
<tr>
<td>show template interface source built-in all command</td>
<td>221</td>
</tr>
<tr>
<td>show template interface source built-in IP_PHONE_INTERFACE_TEMPLATE command</td>
<td>221</td>
</tr>
<tr>
<td>shutdown command</td>
<td>247</td>
</tr>
<tr>
<td>silicon transistors</td>
<td>68</td>
</tr>
<tr>
<td>SimpleAciUiLogServer</td>
<td>208</td>
</tr>
<tr>
<td>SimpleMath.py</td>
<td>36</td>
</tr>
<tr>
<td>simplified networking</td>
<td>4</td>
</tr>
<tr>
<td>Smart Call Home</td>
<td></td>
</tr>
<tr>
<td>enabling on Cisco Catalyst Switch</td>
<td>237–243</td>
</tr>
<tr>
<td>overview</td>
<td>236–237</td>
</tr>
<tr>
<td>website</td>
<td>239, 243</td>
</tr>
<tr>
<td>SMI (Cisco Smart Install) Proxy</td>
<td>271</td>
</tr>
<tr>
<td>SMPs (symmetric multiprocessors), NX-OS support for</td>
<td>69</td>
</tr>
<tr>
<td>SNMP (Simple Network Management Protocol)</td>
<td>258</td>
</tr>
<tr>
<td>soaking state (faults)</td>
<td>172</td>
</tr>
<tr>
<td>soaking-clearing state (faults)</td>
<td>172</td>
</tr>
<tr>
<td>software as a service (SaaS)</td>
<td>6–7</td>
</tr>
<tr>
<td>software development</td>
<td></td>
</tr>
<tr>
<td>common constructs</td>
<td></td>
</tr>
<tr>
<td>conditions, 17</td>
<td></td>
</tr>
<tr>
<td>functions, 18–19</td>
<td></td>
</tr>
<tr>
<td>loops, 18</td>
<td></td>
</tr>
<tr>
<td>objects, 19–20</td>
<td></td>
</tr>
<tr>
<td>variables, 15–17</td>
<td></td>
</tr>
<tr>
<td>Linux</td>
<td></td>
</tr>
<tr>
<td>architecture, 58</td>
<td></td>
</tr>
<tr>
<td>commands, 56–58</td>
<td></td>
</tr>
<tr>
<td>directories, 64</td>
<td></td>
</tr>
<tr>
<td>file system, 63–64</td>
<td></td>
</tr>
<tr>
<td>installing applications on, 64</td>
<td></td>
</tr>
<tr>
<td>overview, 55–56</td>
<td></td>
</tr>
<tr>
<td>permissions, 63–64</td>
<td></td>
</tr>
<tr>
<td>processes, displaying, 59–61</td>
<td></td>
</tr>
<tr>
<td>systemd, 61–62</td>
<td></td>
</tr>
<tr>
<td>overview, 13–15</td>
<td></td>
</tr>
<tr>
<td>pseudocode, 14</td>
<td></td>
</tr>
<tr>
<td>with Python, See Python</td>
<td></td>
</tr>
<tr>
<td>resources, 65–66</td>
<td></td>
</tr>
<tr>
<td>return on investment (ROI), 13–14</td>
<td></td>
</tr>
<tr>
<td>source code, editing</td>
<td></td>
</tr>
<tr>
<td>overview, 49–50</td>
<td></td>
</tr>
<tr>
<td>PyCharm, 50–55</td>
<td></td>
</tr>
<tr>
<td>version control with Git/GitHub</td>
<td></td>
</tr>
<tr>
<td>Git commands, 49</td>
<td></td>
</tr>
<tr>
<td>overview, 45</td>
<td></td>
</tr>
<tr>
<td>repository creation, 45–47</td>
<td></td>
</tr>
<tr>
<td>repository updates, 47–48</td>
<td></td>
</tr>
<tr>
<td>software development kits (SDKs), 37</td>
<td></td>
</tr>
<tr>
<td>Software Images tab (APIC-EM PnP)</td>
<td>274–275</td>
</tr>
</tbody>
</table>
software-defined networking (SDN), 8–9, 304
software-defined WAN (SD-WAN), 264–269

source code, editing
  overview, 49–50
  PyCharm
    debugging, 54–55
    interface, 50–53
    writing code in, 53

source command, 245
SPAN (Switched Port Analyzer), 74
spanning tree protocol (STP), 78
spine/leaf architecture, 76–77
split method, 22
splitting strings, 22
square brackets ([ ]), 32, 33
startup, running Bash scripts at, 103–106
stateless access control lists (ACLs), 79

statements
  case, 92–93
  else, 17, 24–25
  if
    Bash scripting, 91–92
    overview, 17
    Python, 24–25
  if-else
    overview, 17
    Python, 24–25
  traceback statements, 167
states (faults), 171–173
status of interfaces, checking, 101
STP (spanning tree protocol), 78
str command, 23
stream editor (Sed), 96–98

strings
  concatenating, 23
  converting variables to, 23
  overview, 16
  Python, 22–23
  searching, 22
  splitting, 22

su command, 57

subscriptions
  Cisco ACI event subscription, 196–198
  NX-API event subscription, 143–146
  SUDI (secure unique device identification), 270

SWITCH_INTERFACE_TEMPLATE, 220

Switched Port Analyzer (SPAN), 74
switches. See Cisco Catalyst Switch, enabling automation tools on
symmetric multiprocessors (SMPs), NX-OS support for, 69

sys library, 30, 34–35

system events, 112
systemd, 61–62

system-wide health scores, 174

T

T2 network forwarding engine, 67
tail command, 57, 58
tar command, 57
Tcl scripts, EEM (Embedded Event Manager) and, 251–252
Tcl Shell, 243–246
tcpdump, 101–103

templates, AutoConf
interface templates and descriptions, 220–222

template modification, 224–227
tenant configuration (ACI), 204–207
tenant health scores, 174
testing POAP (power-on auto provisioning) files, 87
text, editing in Bash
  Awk, 98–99
  Sed, 96–98
Ticket API
  Postman API, 288–296
  users, viewing, 292–293
ASP (AutoSmart Ports)
  availability, 216
  enabling on Cisco Catalyst Switch, 217–219
Auto Security
  availability, 230
  enabling on Cisco Catalyst Switch, 228–230
  overview, 227–228
AutoConf
  availability, 220
  enabling on Cisco Catalyst Switch, 222–224
  interface templates and descriptions, 220–222
  template modification, 224–227
  website, 227
AutoQoS on LAN devices
  enabling on Cisco Catalyst Switch, 231–233
  overview, 230–231
AutoQoS on WAN devices
  enabling on Cisco ISR routers, 234–236
  overview, 230–231, 233–234
ConfD, 259–263
device-specific macros and descriptions, 217
NETCONF (network configuration protocol), 258–259
overview, 216–217, 255–256
YANG (yet another next generation) data models, 256–258
top command, 61
top of rack (TOR) switches, 75
Topology application (APIC-EM), 279–281
TOR (top of rack) switches, 75
touch command, 47, 57
TP_INTERFACE_TEMPLATE, 220
traceback statements, 167
tracing path, 276–278
traffic classification, 285
transactions (NETCONF), 259
transistors, 68
transport gateways, 236–237
Trident 2 network forwarding engine (NFE), 67
tuples, 27
types, 15–16, 27–28

V
validation (ConfD), 262
variables
Bash scripting, 89–90
EEM (Embedded Event Manager), 113
Python variables
  converting to strings, 23
defining, 15–16
explained, 20–21
bases, 17
lists, 16
names, 16
passing to functions, 19
strings, 16, 22–23
/var/log directory, 64
varying cash flows, 251
verifying ciscopuppet, 154
version control with Git/GitHub
  Git commands, 49
  overview, 45
  repository creation, 45–47
  repository updates, 47–48
View Small button (Path Trace application), 278
viewing
  APIC-EM users, 294–296
  Linux processes, 59–61
virtual environments
  accessing, 163–164
  creating, 163
  customizing, 163
  exiting, 163–164
  freezing, 164–165
  installing libraries into, 164
  overview, 163–165
  in PyCharm, 166

U
UCS (Unified Computing System) Manager, 7
UCS Director (UCS-D), 211
UCS-D (UCS Director), 211
Unicode, 44
Unified Computing System (UCS) Manager, 7
Unplanned Devices screen (APIC-EM PnP), 275–276
updating GitHub repositories, 47–48
USC-Director, 7
use cases (NDB), 75–76
User API
  available APIs, 296
  Postman API, 294–296
users, viewing, 294–296
/usr directory, 64
recreating, 165
sharing, 165
Virtual Extensible LANs (VXLANs), 79–80
Virtual Topology System (VTS), 81
virtualenv
installing, 162–163
in PyCharm, 166
virtual environments, 163–165
virtual-service commands, 107
Visore, 134–136, 182–185
vsh command, 99–100
VTS (Virtual Topology System), 81
VXLANs (Virtual Extensible LANs), 79–80

W

WAN, IWAN (Intelligent WAN), 264–269
WANs, AutoQoS on WAN devices
enabling on Cisco ISR routers, 234–236
overview, 233–234
warning faults, 173
weather, checking, 53
web technologies
Google Postman, 40–43
JSON (Javascript Object Notation), 39–45
REST (Representational State Transfer), 37–38
XML (Extensible Markup Language), 38–39
WebArya, 211
websites
APIC-EM (application policy infrastructure controller enterprise module), 278
ASP (AutoSmart Ports), 220
AutoConf, 227
Cisco DevNet, 204
Cisco IWAN (Intelligent WAN), 269
EEM (Embedded Event Manager), 253
jsoneditoronline.org, 139
NFM (Nexus Fabric Manager), 82
QoS (Quality of Service), 230
RESTful API, 75
Smart Call Home, 239, 243
VTS (Virtual Topology System), 82
WebSoc.py, 144–145
while loops
Bash scripting, 94
overview, 18, 27–28
in Python, 26–27
Wind River, 69
wireless LAN controller (WLC), 277
WLC (wireless LAN controller), 277
write operations (Cisco ACI), 177

X-Y-Z

XML (Extensible Markup Language), 38–39
YAML (yet another mark-up language), 156
YANG (yet another next generation) data models, 256–258
yet another mark-up language (YAML), 156
yet another next generation (YANG) data models, 256–258
Yocto project, 69
yum tool, 65