



The Policy Driven Data Center with ACI

Architecture, Concepts, and Methodology

ciscopress.com

Lucien Avramov, CCIE No. 19945
Maurizio Portolani

FREE SAMPLE CHAPTER



SHARE WITH OTHERS

The Policy Driven Data Center with ACI: Architecture, Concepts, and Methodology

Lucien Avramov, CCIE No. 19945

Maurizio Portolani

Cisco Press

800 East 96th Street

Indianapolis, IN 46240

The Policy Driven Data Center with ACI: Architecture, Concepts, and Methodology

Lucien Avramov and Maurizio Portolani

Copyright © 2015 Cisco Systems, Inc.

Cisco Press logo is a trademark of Cisco Systems, Inc.

Published by:

Cisco Press

800 East 96th Street

Indianapolis, IN 46240 USA

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

Printed in the United States of America

Third Printing: February 2015

Library of Congress Control Number: 2014955987

ISBN-13: 978-1-58714-490-5

ISBN-10: 1-58714-490-5

Warning and Disclaimer

This book is designed to provide information about Cisco ACI. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an “as is” basis. The authors, Cisco Press, and Cisco Systems, Inc. shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the discs or programs that may accompany it.

The opinions expressed in this book belong to the author and are not necessarily those of Cisco Systems, Inc.

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.

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Cisco Press or Cisco Systems, Inc. cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Publisher: Paul Boger

Associate Publisher: Dave Dusthimer

Business Operation Manager, Cisco Press: Jan Cornelssen

Executive Editor: Brett Bartow

Managing Editor: Sandra Schroeder

Development Editor: Marianne Bartow

Project Editor: Mandie Frank

Copy Editor: Bill McManus

Technical Editors: Tom Edsall, Mike Cohen, Krishna Doddapaneni

Editorial Assistant: Vanessa Evans

Designer: Mark Shirar

Composition: Bumpy Design

Indexer: Cheryl Lenser

Proofreader: Debbie Williams



Americas Headquarters
Cisco Systems, Inc.
San Jose, CA

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

Europe Headquarters
Cisco Systems International BV
Amsterdam, The Netherlands

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

CCDE, CCENT, Cisco Eos, Cisco HealthPresence, the Cisco logo, Cisco Lumin, Cisco Nexus, Cisco StadiumVision, Cisco TelePresence, Cisco WebEx, DCE, and Welcome to the Human Network are trademarks. Changing the Way We Work, Live, Play, and Learn and Cisco Store are service marks, and Access Registrar, Aironet, AsyncOS, Bringing the Meeting To You, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, CCSP, CCVP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Collaboration Without Limitation, EtherFast, EtherSwitch, Event Center, Fast Step, Follow Me Browsing, FormShare, GigaDrive, HomeLink, Internet Quotient, IOS, iPhone, iQuick Study, IronPort, the IronPort logo, LightStream, Linksys, MediaTone, MeetingPlace, MeetingPlace Chime Sound, MGX, Networkers, Networking Academy, Network Registrar, PCNow, PIX, PowerPanels, ProConnect, ScriptShare, SenderBase, SMARTnet, Spectrum Expert, StackWise, The Fastest Way to Increase Your Internet Quotient, TransPath, WebEx, and the WebEx logo are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0812R)

About the Authors

Lucien Avramov, CCIE 19945, is a Senior Technical Marketing Engineer at Cisco. Lucien specializes in the Nexus data center portfolio and the ACI. Lucien designs data-center networks worldwide and has wide experience in switch architectures, QoS, ultra-low latency networks, high-performance computing designs, and OpenStack. Lucien is a distinguished Cisco Live speaker and former TAC technical leader, he has several industry certifications, authors RFCs at IETF, and owns an active patent. Lucien holds a master's degree in Computer Science and a bachelor's degree in General Engineering from Ecole des Mines d'Ales, France. In his spare time, Lucien can be found hiking, biking, running marathons around the world, and on Twitter: @flying91.

Maurizio Portolani, Distinguished Technical Marketing Engineer at Cisco Systems, focuses on the design of data center networks. He coauthored *Data Center Fundamentals* for Cisco Press, and holds several patents on current data center technologies. He attended the Politecnico of Torino ("Laurea in Ingegneria") and Ecole Centrale Paris ("Diplôme d'Ingénieur") where he majored in Electronics.

About the Technical Reviewers

Tom Edsall is the Chief Technology Officer of Cisco's Insieme Business Unit, a Cisco Fellow, and a co-founder of Insieme Networks, a developer of application-centric infrastructure products, where he is responsible for system architecture and product evangelism. Insieme Networks was described in *Network World* as "one of the most anticipated events in the networking industry over the past 18 months or so, ever since word leaked that Cisco was funding the spin-in as its response to the software-defined networking trend." At Insieme (recently spun back into Cisco), Edsall has led the development of the Application Centric Infrastructure (ACI), which includes a new line of Nexus 9000 switches that form an application-aware switching fabric along with a centralized controller that manages both virtual and physical network infrastructures.

Tom has been with Cisco since 1993, except for a stint as CTO and co-founder of spin-in Andiamo Systems (building SAN switches). One of Cisco's leading switch architects, he has been responsible for the MDS, Nexus 7000, and Catalyst 5000 and 6000 product lines. Two of his products, the Catalyst 6000 and Nexus 7000, have been the recipients of the prestigious Cisco Pioneer Award. During this time he has been awarded more than 70 patents in the networking industry and was recently an author of "CONGA: Distributed Congestion-Aware Load Balancing for Data Centers," which won the prestigious SIGCOMM 2014 best paper award.

Before joining Cisco, Tom was a co-founder and a member of the senior engineering management team at Crescendo Communications, Cisco's first acquisition. Edsall holds BSEE and MSEE degrees from Stanford, where he has also been a Visiting Scholar and occasional lecturer.

Mike Cohen is Director of Product Management at Cisco Systems. Mike began his career as an early engineer on VMware's hypervisor team and subsequently worked in infrastructure product management on Google and Big Switch Networks. Mike holds a BSE in Electrical Engineering from Princeton University and an MBA from Harvard Business School.

Krishna Doddapaneni is responsible for the switching infrastructure and iNXOS part of ACI. Previously, he served as Director in SAVBU (Cisco) (part of the acquisition of Nuova Systems). In Nuova Systems, he was responsible for the delivery of the first FCoE switch. He was responsible for multiple generations of Nexus 5k/2k product lines. Before Nuova, he was the first employee of Greenfield Networks (acquired by Cisco). He holds an MS degree in Computer Engineering from Texas A&M University. He holds numerous patents in the networking field.

Dedications

Lucien Avramov:

For Regina and Michel, my precious parents who made lifetime sacrifices to give me a better future.

Maurizio Portolani:

This book is dedicated to my friends and my family.

Acknowledgments

We would like to thank Mike Dvorkin, Tom Edsall, and Praveen Jain for founding ACI.

Lucien Avramov:

First, I would like to thank my family, friends, colleagues, customers, and mentors for supporting me during this journey, you know who you are. It means a lot to me. Thank you Mike Dvorkin for sharing your knowledge, philosophy, and friendship. Mike Cohen, thank you for always being available and willing to work hard on the reviews, your opinions, and being a great friend. Tom Edsall, thank you for the quality feedback and time you gave us in this project. Takashi Oikawa, thank you for your kindness and wisdom. Along this journey I made friends and shared incredible memories. Writing a book is a journey mainly with yourself, with a reward comparable to reaching a summit. This journey is stronger when shared with a co-author: I am fortunate to have made a great friend along the way, Maurizio Portolani.

Second, I thank Ron Fuller for introducing me to the pleasure of going into a book project. Thank you to my Cisco colleagues who supported me along the way: Francois Couderc for the great knowledge sharing, time spent thinking about the structure of this book, your advice and reviews; Chih-Tsung Huang, Garry Lemasa, Arkadiy Shapiro, Mike Pavlovich, Jonathan Cornell, and Aleksandr Oysgelt for your encouragement, reviews, and support along the way. A profound acknowledgement and thanks to the Cisco Press team: Brett Bartow, your kindness, availability, and patience have meant a lot to me. Thank you for the opportunity to develop this content and for giving me a chance. Marianne Bartow, thank you for spending so much time with quality reviews. Bill McManus, thank you for the editing. Chris Cleveland, thank you for your support along the way. Mandie Frank, thank you for all the efforts, including keeping this project on time; and Mark Shirar, for design help.

Finally, I thank the people who gave me a chance in my professional career, starting with Jean-Louis Delhaye mentoring me for years at Airbus and being my friend ever since, Didier Fernandes for introducing me and mentoring me in Cisco, Erin Foster for giving me a chance to join Cisco and relocating me to the United States, Ed Swenson and Ali Ali for giving me a full time job in Cisco TAC, John Bunney for taking me along to build the TAC Data Center team and mentoring me. Thank you Yousuf Khan for giving me a chance to join Technical Marketing first, in the Nexus Team, and later in the ACI team, and for coaching me along the way; Jacob Rapp, Pramod Srivatsa, and Tuqiang Cao for your leadership and developing my career.

Maurizio Portolani:

I would personally like to acknowledge many people who opened my eyes to modern software development methodology and technology that I could relate to the changes that ACI is bringing to networking. A special acknowledgment goes to Marco Molteni for his in-depth philosophical views on XML versus JSON and Yaml and for enlightening me on GitHub and Python. I would also like to acknowledge Amine Choukir in particular for his insights on continuous integration, and Luca Relandini for his expertise on automation.

Contents at a Glance

	Foreword	xx
	Introduction	xxi
Chapter 1	Data Center Architecture Considerations	1
Chapter 2	Building Blocks for Cloud Architectures	37
Chapter 3	The Policy Data Center	57
Chapter 4	Operational Model	91
Chapter 5	Data Center Design with Hypervisors	127
Chapter 6	OpenStack	167
Chapter 7	ACI Fabric Design Methodology	193
Chapter 8	Service Insertion with ACI	243
Chapter 9	Advanced Telemetry	267
Chapter 10	Data Center Switch Architecture	285
	Conclusion	329
	Index	331

Contents

Foreword xx
Introduction xxi

Chapter 1 Data Center Architecture Considerations 1

Application and Storage	1
Virtualized Data Center	2
<i>Introduction</i>	2
<i>Definition and Virtualization Concepts</i>	3
<i>Network and Design Requirements</i>	6
<i>Storage Requirements</i>	7
Big Data	7
<i>Definition</i>	7
<i>Network Requirements</i>	9
<i>Cluster Design with the Hadoop Building Blocks: the POD</i>	10
<i>Storage Requirements</i>	11
<i>Design Considerations</i>	11
High-Performance Compute	14
<i>Definition</i>	14
<i>Network Requirements</i>	14
<i>Storage Requirements</i>	14
<i>Design Considerations</i>	14
<i>Design Topologies</i>	15
Ultra-Low Latency	16
<i>Definition</i>	16
<i>Network Requirements</i>	17
<i>Storage Requirements</i>	18
<i>Design Considerations</i>	18
<i>Design Topologies</i>	19
Massively Scalable Data Center	21
<i>Definition</i>	21
<i>Network Requirements</i>	23
<i>Storage Requirements</i>	24
<i>Design Considerations</i>	24
<i>Design Topologies</i>	25
Design Topologies Examples	25

The POD-based Designs	26
The POD Model or the Data Model for Shared Infrastructure and Cloud Computing	26
The FlexPod Design	28
Data Center Designs	29
End of Row	29
<i>Middle of Row</i>	30
<i>Top of Rack: The Modern Data Center Approach</i>	30
<i>Single-Homed Servers Design</i>	32
Logical Data Center Design with the Spine-Leaf ACI Foundation Architecture	33
Summary	35

Chapter 2 Building Blocks for Cloud Architectures 37

Introduction to Cloud Architectures	37
Network Requirements of Clouds and the ACI Solution	39
Amazon Web Services Model	41
Automating Server Provisioning	43
PXE Booting	43
Deploying the OS with Chef, Puppet, CFengine, or Similar Tools	44
<i>Chef</i>	45
<i>Puppet</i>	46
Orchestrators for Infrastructure as a Service	47
vCloud Director	47
OpenStack	48
<i>Project and Releases</i>	48
<i>Multi-Hypervisor Support</i>	49
<i>Installers</i>	49
<i>Architecture Models</i>	50
<i>Networking Considerations</i>	51
UCS Director	51
Cisco Intelligent Automation for Cloud	52
Conciliating Different Abstraction Models	55
Summary	56

Chapter 3 The Policy Data Center 57

Why the Need for the Policy-Based Model?	57
The Policy Theory	59
Cisco APIC Policy Object Model	61
Endpoint Groups	63
Cisco APIC Policy Enforcement	66
<i>Unicast Policy Enforcement</i>	66
<i>Multicast Policy Enforcement</i>	69
Application Network Profiles	70
Contracts	71
Understanding Cisco APIC	79
Cisco ACI Operating System (Cisco ACI Fabric OS)	79
Architecture: Components and Functions of the Cisco APIC	80
Policy Manager	81
Topology Manager	81
Observer	82
Boot Director	82
Appliance Director	83
VMM Manager	83
Event Manager	83
Appliance Element	84
Architecture: Data Management with Sharding	84
<i>Effect of Replication on Reliability</i>	84
<i>Effect of Sharding on Reliability</i>	85
<i>Sharding Technology</i>	86
User Interface: Graphical User Interface	87
User Interface: Command-Line Interface	87
User Interface: RESTful API	88
System Access: Authentication, Authorization, and RBAC	88
Summary	89

Chapter 4 Operational Model 91

Introduction to Key Technologies and Tools for Modern Data Centers	92
Network Management Options	92
REST Protocol	93
XML, JSON, and YAML	94

Python	96
<i>Python Basics</i>	96
<i>Where Is the main() Function?</i>	97
<i>Functions Definition</i>	97
<i>Useful Data Structures</i>	98
<i>Parsing Files</i>	99
<i>Verifying Python Scripts</i>	101
<i>Where to Run Python</i>	101
<i>Pip, EasyInstall, and Setup Tools</i>	101
<i>Which Packages Do I Need?</i>	101
<i>virtualenv</i>	102
Git and GitHub	103
<i>Basic Concepts of Version Control</i>	103
<i>Centralized Versus Distributed</i>	104
<i>Overview of Basic Operations with Git</i>	104
<i>Installing/Setting Up Git</i>	105
<i>Key Commands in Git</i>	105
Operations with the Cisco APIC	106
Object Tree	108
<i>Classes, Objects, and Relations</i>	109
<i>Naming Conventions</i>	113
<i>Object Store</i>	114
Using REST to Program the Network	114
<i>Tools to Send REST Calls</i>	115
<i>REST Syntax in Cisco ACI</i>	117
<i>Modeling Tenants in XML</i>	119
<i>Defining the Relationship Among EPGs (Providers and Consumers)</i>	120
<i>A Simple Any-to-Any Policy</i>	121
ACI SDK	122
<i>ACI Python Egg</i>	122
<i>How to Develop Python Scripts for ACI</i>	123
<i>Where to Find Python Scripts for ACI</i>	124
For Additional Information	124
Summary	125

Chapter 5 Data Center Design with Hypervisors 127

Virtualized Server Networking	128
Why Have a Software Switching Component on the Server?	129
Overview of Networking Components	132
<i>Virtual Network Adapters</i>	132
<i>Virtual Switching</i>	133
<i>Endpoint Groups</i>	133
<i>Distributed Switching</i>	133
Hot Migration of Virtual Machines	134
Segmentation Options	134
VLANs	134
VXLANs	134
<i>VXLAN Packet Format</i>	135
<i>VXLAN Packet Forwarding</i>	136
<i>VXLANs Without Multicast</i>	137
Microsoft Hyper-V Networking	137
Linux KVM and Networking	141
Linux Bridging	142
Open vSwitch	143
<i>OVS Architecture</i>	143
<i>Example Topology</i>	145
<i>Open vSwitch with OpenStack</i>	146
<i>OpenFlow</i>	147
VMware ESX/ESXi Networking	149
VMware vSwitch and Distributed Virtual Switch	150
VMware ESXi Server Traffic Requirements	151
<i>VXLAN Tagging with vShield</i>	151
vCloud Director and vApps	152
<i>vCloud Networks</i>	153
Cisco Nexus 1000V	155
Port Extension with VN-TAG	158
Cisco ACI Modeling of Virtual Server Connectivity	160
Overlay Normalization	160
VMM Domain	161
Endpoint Discovery	162
Policy Resolution Immediacy	162

- Cisco ACI Integration with Hyper-V 162
- Cisco ACI Integration with KVM 163
- Cisco ACI Integration with VMware ESX 164
- Summary 165

Chapter 6 OpenStack 167

- What Is OpenStack? 167
 - Nova 168
 - Neutron 169
 - Swift 173
 - Cinder 173
 - Horizon 174
 - Heat 174
 - Ironic 174
- OpenStack Deployments in the Enterprise 176
- Benefits of Cisco ACI and OpenStack 177
 - Cisco ACI Policy Model 178
 - Physical and Virtual Integration 179
 - Fabric Tunnels 179
 - Service Chaining 179
 - Telemetry 179
- OpenStack APIC Driver Architecture and Operations 180
 - How Integration Works 180
- Deployment Example 182
 - Installation of Icehouse 183
 - Configuration of the Cisco APIC Driver 185
 - Neutron.conf* File 186
 - ML2_conf.ini* File 186
 - ML2_cisco_conf.ini* File 186
 - Configuration Parameters* 187
 - Host-Port Connectivity* 188
 - External Networks* 188
 - PortChannel Configuration* 188
 - Troubleshooting 188
- The Group Based Policy Project at OpenStack 190
- Summary 191

Chapter 7	ACI Fabric Design Methodology	193
	Summary of ACI Fabric Key Functionalities	194
	ACI Forwarding Behavior	194
	<i>Prescriptive Topology</i>	194
	<i>Overlay Frame Format</i>	196
	VXLAN Forwarding	197
	<i>Pervasive Gateway</i>	198
	<i>Outside Versus Inside</i>	199
	<i>Packet Walk</i>	201
	Segmentation with Endpoint Groups	202
	Management Model	204
	Hardware and Software	207
	Physical Topology	208
	Cisco APIC Design Considerations	210
	Spine Design Considerations	211
	Leaf Design Considerations	212
	<i>Unknown Unicast and Broadcast</i>	213
	<i>Use of VLANs as a Segmentation Mechanism</i>	214
	VLANs and VXLANs Namespaces	215
	<i>Concept of Domain</i>	216
	<i>Concept of Attach Entry Profile</i>	217
	Multi-tenancy Considerations	218
	Initial Configuration Steps	219
	Zero-Touch Provisioning	220
	Network Management	221
	Policy-based Configuration of Access Ports	223
	<i>Configuring Switch Profiles for Each Leaf</i>	228
	<i>Configuring Interface Policies</i>	228
	Interface Policy Groups and PortChannels	228
	<i>Interface Policy Groups</i>	229
	<i>PortChannels</i>	229
	<i>Virtual PortChannels</i>	231
	Virtual Machine Manager (VMM) Domains	233
	VMM Domain	233
	AEP for Virtualized Servers Connectivity	234

Configuring a Virtual Topology	235
Bridge Domain	237
<i>Hardware Proxy</i>	237
<i>Flooding Mode</i>	238
<i>fvCtx</i>	238
Endpoint Connectivity	238
<i>Connecting a Physical Server</i>	239
<i>Connecting a Virtual Server</i>	239
External Connectivity	240
Summary	241

Chapter 8 Service Insertion with ACI 243

Overview of ACI Design with Layer 4 Through Layer 7 Services	244
Benefits	244
Connecting Endpoint Groups with a Service Graph	244
Extension to Virtualized Servers	245
Management Model	245
Service Graphs, Functions, and Rendering	246
Hardware and Software Support	247
Cisco ACI Modeling of Service Insertion	248
Service Graph Definition	249
Concrete Devices and Logical Devices	250
Logical Device Selector (or Context)	251
Splitting Bridge Domains	251
Configuration Steps	252
Definition of a Service Graph	253
<i>Defining the Boundaries of the Service Graph</i>	253
<i>The Metadevice</i>	254
<i>Defining an Abstract Node's Functions</i>	255
<i>Defining an Abstract Node's Connectors</i>	257
<i>Abstract Node Elements Summary</i>	258
<i>Connecting Abstract Nodes to Create the Graph</i>	258
Definition of Concrete Devices and Cluster of Concrete Devices	260
<i>Configuration of the Logical Device and Concrete Device</i>	261
<i>Configuration of the Logical Device Context (Cluster Device Selector)</i>	264
<i>Naming Summary</i>	265
Summary	266

Chapter 9 Advanced Telemetry 267

- Atomic Counters 267
 - The Principle 267
 - Further Explanation and Example 268
 - Atomic Counters and the APIC 270
- Latency Metrics 271
- ACI Health Monitoring 272
 - Statistics 273
 - Faults 274
 - Events, Logs, Diagnostics 279
 - Health Score 280
- The Centralized show tech-support ACI Approach 281
- Summary 282

Chapter 10 Data Center Switch Architecture 285

- Data, Control, and Management Planes 285
 - Separation Between Data, Control, and Management Planes 286
 - Interaction Between Control, Data, and Management Planes 287
 - Protection of the Control Plane with CoPP 288
 - Control Plane Packet Types* 288
 - CoPP Classification* 290
 - CoPP Rate-Controlling Mechanisms* 290
- Data Center Switch Architecture 291
 - Cut-through Switching: Performance for the Data Center 292
 - Crossbar Switch Fabric Architecture 295
 - Unicast Switching over Crossbar Fabrics* 297
 - Multicast Switching over Crossbar Fabrics* 298
 - Overspeed in Crossbar Fabrics* 298
 - Superframing in the Crossbar Fabric* 299
 - The Scheduler* 301
 - Crossbar Cut-through Architecture Summary* 301
 - Output Queuing (Classic Crossbar)* 302
 - Input Queuing (Ingress Crossbar)* 303
 - Understanding HOLB* 304
 - Overcoming HOLB with VoQ* 304
 - Multistage Crossbar* 305
 - Centralized Shared Memory (SoC) 306

Multistage SoC	307
<i>Crossbar Fabric with SoC</i>	307
<i>SoC Fabric</i>	308
QoS Fundamentals	309
Data Center QoS Requirements	309
<i>Data Center Requirements</i>	311
<i>Type of QoS Used in Different Data Center Use Cases</i>	312
<i>Trust, Classification, and Marking Boundaries</i>	313
Data Center QoS Capabilities	315
<i>Understanding Buffer Utilization</i>	315
<i>The Buffer Bloat</i>	317
<i>Priority Flow Control</i>	318
<i>Enhanced Transmission Selection</i>	319
<i>Data Center Bridging Exchange</i>	320
<i>ECN and DCTCP</i>	320
<i>Priority Queue</i>	321
<i>Flowlet Switching: Nexus 9000 Fabric Load Balancing</i>	322
Nexus QoS Implementation: The MQC Model	324
Summary	326
Conclusion	329
Index	331

Command Syntax Conventions

The conventions used to present command syntax in this book are the same conventions used in Cisco's 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).
- *Italics* indicate arguments for which you supply actual values.
- Vertical bars (|) separate alternative, mutually exclusive elements.
- Square brackets [] indicate optional elements.
- Braces { } indicate a required choice.
- Braces within brackets [{ }] indicate a required choice within an optional element.

Note This book covers multiple operating systems, and different icons and router names are used to indicate the appropriate OS that is being referenced. Cisco IOS and IOS XE use router names such as R1 and R2 and are referenced by the IOS router icon. Cisco IOS XR routers use router names such as XR1 and XR2 and are referenced by the IOS XR router icon.

Foreword

Looking at the history of network control, one can wonder why so much complexity emerged out of so simple concepts. Network management systems have traditionally focused on control of features, without thinking of networks as systems. Any network control scheme, at the heart, aims to solve two things: control of endpoint behaviors, where regulations are imposed on what sets of endpoints can communicate or not, also known as access control, and path optimization problems instrumented through management of numerous network control plane protocols. Unfortunately, this natural separation has rarely been honored, resulting in the control models that are both difficult to consume and operationally fragile.

IT does not exist for the benefit of itself. The purpose of any IT organization is to run business applications. The application owner, architect, and developer all have intimate understanding of their applications. They have a complete picture of the application's infrastructure requirements and full understanding of other application components necessary for communication. However, once it comes to deployment, all this knowledge, the original intent, is forever lost in the implementation detail of the translation between the application requirements and the actual configuration of the infrastructure. The unfortunate consequence of this is that there's no easy way to map resources and configurations back to the application. Now, what if we need to expand the app, add more components, or simply retire it from the data center? What happens to the residual configuration?

When we started Insieme, one of the chief goals was to bring networking into the reach of those who don't need to understand it: an application guy who needs to identify how his application interacts with other application components in the data center, an ops guy who needs to configure cluster expansion, a compliance guy who needs to ensure that no enterprise-wide business rules are violated. We felt that the way operational teams interact with the network needed to change in order for networking to enter the next logical step in the evolution.

Lucien and Maurizio explain the new Policy Driven Data Center and its associated operational model. This book focuses, on one hand, on the architecture, concept, and methodology to build a modern data center solving this paradigm; while also, on the other hand, detailing the Cisco ACI solution.

Mike Dvorkin

Distinguished Cisco Engineer, Chief Scientist, and Co-founder of Insieme Networks

Introduction

Welcome to the Policy Driven Data Center with Application Centric Infrastructure (ACI). You are embarking on a journey to understand the latest Cisco data center fabric and the many innovations that are part of it.

The objective of this book is to explain the architecture design principles, the concepts, and the methodology to build new data center fabrics. Several key concepts in this book, such as the policy data model, programming, and automation, have a domain of applicability that goes beyond the ACI technology itself and forms a core skillset of network engineers and architects.

Cisco Application Centric Infrastructure (ACI) is a data center fabric that enables you to integrate virtual and physical workloads in a highly programmable multi-hypervisor environment that is designed for any multi-service or cloud data center.

To fully appreciate the ACI innovations, one has to understand the key new industry trends in the networking field.

Industry Trends

At the time of this writing, the network industry is experiencing the emergence of new operational models. Many of these changes are influenced by innovations and methodology that have happened in the server world or in the application world.

The following list provides a nonexhaustive collection of trends currently influencing new data center designs:

- Adoption of cloud services.
- New methodology of provisioning network connectivity (namely self-service catalogs).
- Ability to put new applications into production more quickly and to do A/B testing. This concept relates to the ability to shorten the time necessary to provision a complete network infrastructure for a given application.
- Ability to “fail fast”; that is, being able to put a new version of an application into production for a limited time and then to decommission it quickly should bugs arise during the testing.
- Ability to use the same tools that manage servers (such as Puppet, Chef, CFEngines, etc.) to manage networking equipment.
- The need for better interaction between server and application teams and operation teams (DevOps).
- Ability to deal with “elephant flows”; that is, the ability to have backups or commonly bulk transfers without affecting the rest of the traffic.
- Ability to automate network configuration with a more systematic and less prone to error programmatic way using scripts.

- Adoption of software development methodologies such as Agile and Continuous Integration.

Some of these trends are collectively summarized as “application velocity,” which refers to the ability to shorten the time to bring an application from development to production (and back to testing, if needed) by spawning new servers and network connectivity in a much faster way than before.

What Is an “Application”?

The meaning of “application” varies depending on the context or job role of the person that is using this term. For a networking professional, an application may be a DNS server, a virtualized server, a web server, and so on. For a developer of an online ordering tool, the application is the ordering tool itself, which comprises various servers: presentation servers, databases, and so on. For a middleware professional, an application may be the IBM WebSphere environment, SAP, and so on.

For the purpose of this book, in the context of Cisco ACI, an application refers to a set of networking components that provides connectivity for a given set of workloads. These workloads’ relationship is what ACI calls an “application,” and the relationship is expressed by what ACI calls an application network profile, explained after Figure 1.

Figure 1 provides an example illustrating an application that is accessible from a company intranet and that is connected to an external company that provides some business function. This could be, for instance, a travel reservation system, an ordering tool, a billing tool, and so on.

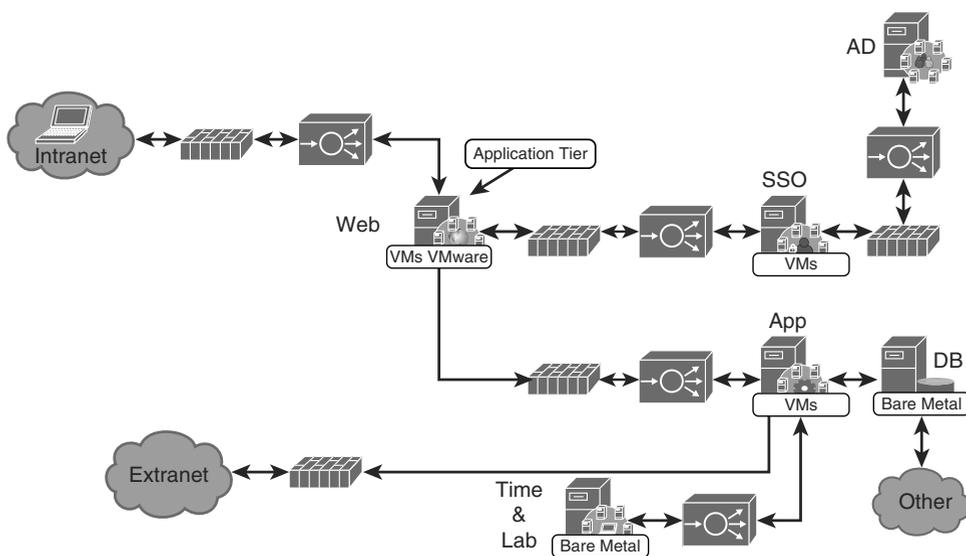


Figure 1 Example of an “Application”

This relationship can be expressed in ACI by using the concept of *application network profile* (ANP), which abstracts the specific VLANs or subnets that the building blocks reside on. The configuration of network connectivity is expressed in terms of *policies*, which define which endpoints consume (or provide) services provided by (consumed by) other endpoints.

Using ACI doesn't require deep understanding of these application relationships. These often are implicit in existing networking configurations by means of VLANs and access control lists. Hence, one can just use ANPs and associated policies as containers of existing configurations without the need to map exact server-to-server communication patterns.

The value proposition of using ANPs is that it enables network administrators to express network configurations in a more abstract manner that can be more closely mapped to the building blocks of a business application such as an ordering tool, a travel reservation system, and so on. After the applications are defined, they can be validated in a test environment and immediately moved to a production environment.

The Need for Abstraction

Applications already run in data centers today even without ACI. Network administrators create the connectivity between building blocks by using VLANs, IP addresses, routing, and ACLs by translating the requirements of the IT organization to support a given tool. However, without ACI, administrators have no way to really express such configurations directly in a format that can be mapped to the network, leaving administrators with no choice but to focus primarily on expressing a very open connectivity policy to ensure that servers can talk to each other if they are internal to the company and can talk to the outside if they are on the DMZ or extranet. This requires administrators to harden ACLs and put firewalls to restrict the scope of which service clients and other servers can use from a given set of servers.

This approach results in configurations that are not very portable. They are very much hard-coded in the specific data center environment where they are implemented. If the same environment must be built in a different data center, somebody must perform the tedious job of reconfiguring IP addresses and VLANs and deciphering ACLs.

ACI is revolutionizing this process by introducing the ability to create an application network profile, a configuration template to express relationships between compute segments. ACI then translates those relationships into networking constructs that routers and switches can implement (i.e., in VLANs, VXLANs, VRFs, IP addresses, and so on).

What Is Cisco ACI

The Cisco ACI fabric consists of discrete components that operate as routers and switches but are provisioned and monitored as a single entity. The operation is like a distributed switch and router configuration that provides advanced traffic optimization, security, and telemetry functions, stitching together virtual and physical workloads. The controller, called the Application Policy Infrastructure Controller (APIC), is the central point of management of the fabric. This is the device that distributes ANP policies to the devices that are part of the fabric.

The Cisco ACI Fabric OS runs on the building blocks of the fabric, which are, at time of writing, the Cisco Nexus 9000 Series nodes. The Cisco ACI Fabric OS is object-oriented and enables programming of objects for each configurable element of the system. The ACI Fabric OS renders policies (such as the ANP and its relationships) from the controller into a concrete model that runs in the physical infrastructure. The concrete model is analogous to compiled software; it is the form of the model that the switch operating system can execute.

Cisco ACI is designed for many types of deployments, including public and private clouds, big data environments, and hosting of virtualized and physical workloads. It provides the ability to instantiate new networks almost instantaneously and to remove them just as quickly. ACI is designed to simplify automation and can be easily integrated into the workflow of common orchestration tools.

Figure 2 illustrates the ACI fabric with the spine-leaf architecture and controllers. Physical and virtual servers can be connected to the ACI fabric and also receive connectivity to the external network.

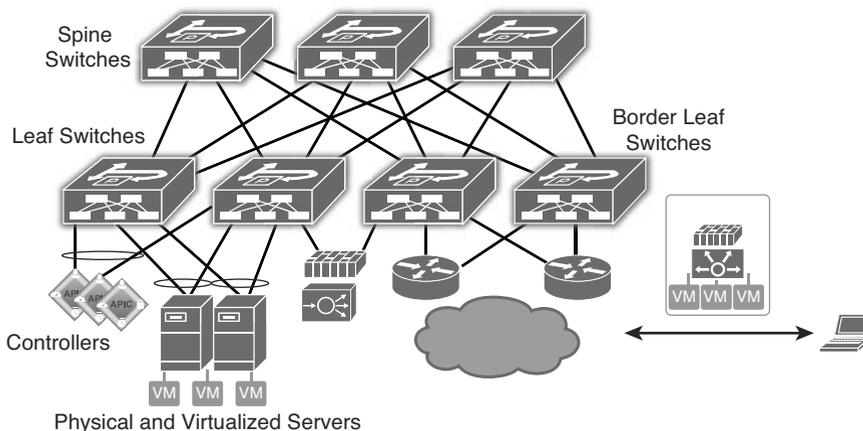


Figure 2 ACI Fabric

Cisco ACI Innovations

Cisco ACI introduces many innovations:

- The whole fabric is managed as a single entity but without a centralized control plane.
- The fabric is managed via an object tree with methods and classes that are accessible with REST calls.
- It introduces a new management model based on a declarative approach instead of an imperative approach.
- It allows a clear mapping of application relationships to the network infrastructure.
- It is designed for multi-tenancy.
- It is multi-hypervisor capable.
- It allows the definition of abstract configurations (or templates) that make configurations portable.
- It changes the way that networking configurations are expressed, from VLAN and IP addresses to policies.
- It revolutionizes equal-cost multipathing and quality of service (QoS) with flowlet load balancing, dynamic flow prioritization, and congestion management.
- It introduces new concepts for telemetry, such as the concept of health scores and atomic counters.

Book Structure

Chapter 1: Data Center Architecture Considerations

The goal of this chapter is to describe the network requirements of different server environments and how to meet them in terms of network design.

Chapter 2: Building Blocks for Cloud Architectures

At the time of this writing, most large-scale data center deployments are designed with the principles of cloud computing. This is equally true for data centers that are built by providers or by large enterprises. This chapter illustrates the design and technology requirements of building a cloud.

Chapter 3: The Policy Data Center

The goal of this chapter is to elucidate the Cisco ACI approach to modeling business applications. This approach provides a unique blend of mapping hardware and software capabilities to the deployment of applications either graphically through the Cisco Application Policy Infrastructure Controller (APIC) GUI or programmatically through

the Cisco APIC API model. The APIC concepts and principles are explained in detail in this chapter. Finally, the ACI fabric is not only for greenfield deployment. Many users will consider how to deploy an ACI fabric into an existing environment. Therefore, the last part of this chapter explains how to integrate the ACI fabric with an existing network.

Chapter 4: Operational Model

Command-line interfaces (CLI) are great tools for interactive changes to the configuration, but they are not designed for automation, nor for ease of parsing (CLI scraping is neither efficient nor practical) or customization. Furthermore, CLIs don't have the ability to compete with the power of parsing, string manipulation, or the advanced logic that sophisticated scripting languages like Python can offer. This chapter covers the key technologies and tools that new administrators and operators must be familiar with, and it explains how they are used in an ACI-based data center.

Chapter 5: Data Center Design with Hypervisors

This chapter describes the networking requirements and design considerations when using hypervisors in the data center.

Chapter 6: OpenStack

This chapter explains in detail OpenStack and its relation to Cisco ACI. The goal of this chapter is to explain what OpenStack is and present the details of the Cisco ACI APIC OpenStack driver architecture.

Chapter 7: ACI Fabric Design Methodology

This chapter describes the topology of an ACI fabric and how to configure it both as an infrastructure administrator and as a tenant administrator. The chapter covers the configuration of physical interfaces, PortChannels, virtual PortChannels, and VLAN namespaces as part of the infrastructure configurations. The chapter also covers the topics of segmentation, multi-tenancy, connectivity to physical and virtual servers, and external connectivity as part of the tenant configuration.

Chapter 8: Service Insertion with ACI

Cisco ACI technology provides the capability to insert Layer 4 through Layer 7 functions using an approach called a service graph. The industry normally refers to the capability to add Layer 4 through Layer 7 devices in the path between endpoints as service insertion. The Cisco ACI service graph technology can be considered a superset of service insertion. This chapter describes the service graph concept and how to design for service insertion with the service graph.

Chapter 9: Advanced Telemetry

The goal of this chapter is to explain the centralized troubleshooting techniques that ACI offers for isolating problems. It includes topics such as atomic counters and health scores.

Chapter 10: Data Center Switch Architecture

The goal of this chapter is to provide a clear explanation of the data center switching architecture. It is divided into three sections: the hardware switch architecture, the fundamental principles of switching, and the quality of service in the data center.

Terminology

Node: Physical network device.

Spine node: Network device placed in the core part of the data center. Typically it's a device with high port density and higher speed.

Leaf node: Network device placed at the access of the data center. It is the first tier of network equipment defining the data center network fabric.

Fabric: A group of leaf and spine nodes defining the data center network physical topology.

Workload: A virtual machine defining a single virtual entity.

Two-tier topology: Typically defined by a spine-leaf fabric topology.

Three-tier topology: A network topology with access, aggregation, and core tiers.

Services: Category defined by the following (nonexhaustive) group of appliances: load balancers, security devices, content accelerators, network monitoring devices, network management devices, traffic analyzers, automation and scripting servers, etc.

ULL: Ultra-low latency. Characterizes network equipment in which the latency is under a microsecond. Current technology is nanosecond level.

HPC: High-performance compute. Applications using structured data schemes (database) or unstructured data (NoSQL) where performance is important at predictable and low latency and with the capability to scale. The traffic patterns are east-west.

HFT: High-frequency trading. Typically occurs in a financial trading environment, where the latency needs to be minimal on the data center fabric to provide as close as possible to real time information to the end users. Traffic is mainly north-south

Clos: Multistage switching network, sometimes called "fat tree," based on a 1985 article by Charles Leiserson. The idea of Clos is to build a very high-speed, nonblocking switching fabric.

This page intentionally left blank

Building Blocks for Cloud Architectures

At the time of this writing, most large-scale data center deployments are designed with the principles of cloud computing at the forefront. This is equally true for data centers that are built by providers or by large enterprises. This chapter illustrates the design and technology requirements for building a cloud.

Introduction to Cloud Architectures

The National Institute of Technology and Standards (NIST) defines cloud computing as “a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” (See <http://csrc.nist.gov/groups/SNS/cloud-computing>.)

Data center resources, such as individual servers or applications, are offered as *elastic* services, which means that capacity is added on demand, and when the compute or application is not needed, the resources providing it can be decommissioned. Amazon Web Services (AWS) is often regarded as the pioneer of this concept and many similar services that exist today.

Cloud computing services are often classified according to two different categories:

- **Cloud delivery model:** Public cloud, private cloud, or hybrid cloud
- **Service delivery model:** Infrastructure as a Service, Platform as a Service, or Software as a Service

The cloud delivery model indicates where the compute is provisioned. The following terminology is often used:

- **Private cloud:** A service on the premises of an enterprise. A data center designed as a private cloud offers shared resources to internal users. A private cloud is shared by tenants, where each tenant is, for instance, a business unit.
- **Public cloud:** A service offered by a service provider or cloud provider such as Amazon, Rackspace, Google, or Microsoft. A public cloud is typically shared by multiple *tenants*, where each tenant is, for instance, an enterprise.
- **Hybrid cloud:** Offers some resources for workloads through a private cloud and other resources through a public cloud. The ability to move some compute to the public cloud is sometimes referred to as *cloud burst*.

The service delivery model indicates what the user employs from the cloud service:

- **Infrastructure as a Service (IaaS):** A user requests a dedicated machine (a virtual machine) on which they install applications, some storage, and networking infrastructure. Examples include Amazon AWS, VMware vCloud Express, and so on.
- **Platform as a Service (PaaS):** A user requests a database, web server environment, and so on. Examples include Google App Engine and Microsoft Azure.
- **Software as a Service (SaaS) or Application as a Service (AaaS):** A user runs applications such as Microsoft Office, Salesforce, or Cisco WebEx on the cloud instead of on their own premises.

The cloud model of consumption of IT services, and in particular for IaaS, is based on the concept that the user relies on a self-service portal to provide services from a catalog and the provisioning workflow is completely automated. This ensures that the user of the service doesn't need to wait for IT personnel to allocate VLANs, stitch load balancers or firewalls, and so on. The key benefit is that the fulfillment of the user's request is quasi-instantaneous.

Until recently, configurations were performed via the CLI to manipulate on a box-by-box basis. Now, ACI offers the ability to instantiate "virtual" networks of a very large scale with a very compact description using Extensible Markup Language (XML) or JavaScript Object Notation (JSON).

Tools such as Cisco UCS Director (UCSD) and Cisco Intelligent Automation for Cloud (CIAC) orchestrate the ACI services together with compute provisioning (such as via Cisco UCS Manager, VMware vCenter, or OpenStack) to provide a fast provisioning service for the entire infrastructure (which the industry terms a *virtual private cloud*, a *virtual data center*, or a *container*).

The components of the cloud infrastructure are represented at a very high level in Figure 2-1. The user (a) of the cloud service (b) orders a self-contained environment (c) represented by the container with firewall load balancing and virtual machines (VM). CIAC

provides the service catalog function, while UCSD and OpenStack operate as the element managers.

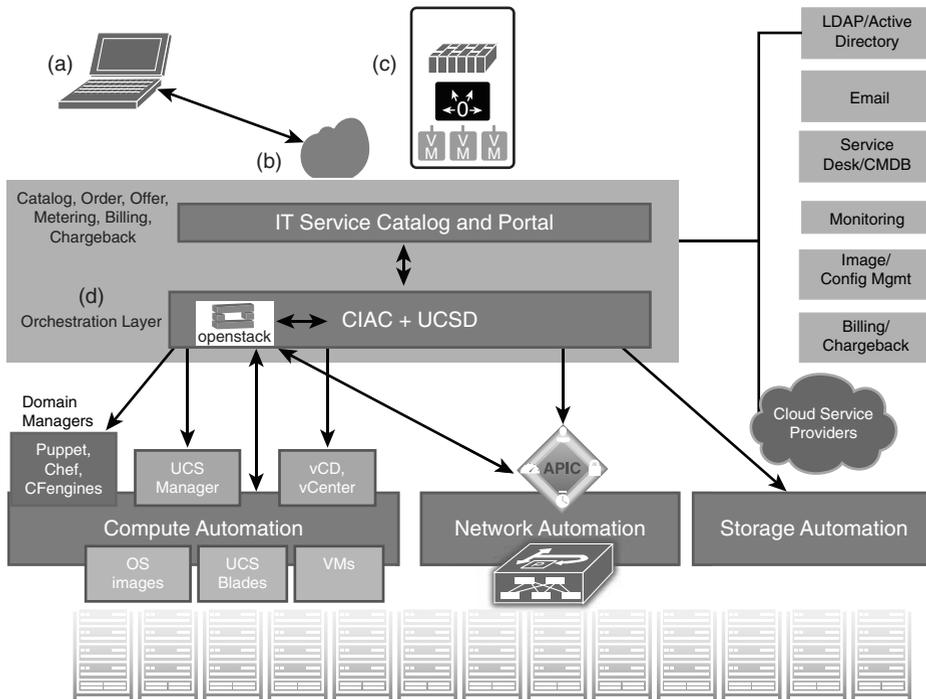


Figure 2-1 *Building Blocks of a Cloud Infrastructure*

This request is serviced by the service catalog and portal via the orchestration layer (d). The orchestration layer can be composed of several components. Cisco, for instance, offers CIAC, which interacts with various element managers to provision compute, network, and storage resources.

Figure 2-1 also explains where Application Centric Infrastructure (ACI) and, more precisely, the Cisco Application Policy Infrastructure Controller (APIC), fit in the cloud architecture.

Network Requirements of Clouds and the ACI Solution

The network infrastructure that provides support for cloud deployments must meet several requirements, such as:

- Scale for a very large number of virtual machines
- Support Layer 2 adjacency between workloads
- Support multi-tenancy

- Be highly programmable
- Support the insertion of load balancers and firewalls
- Support the insertion of virtual load balancers and virtual firewalls

The first and second requirements are almost incompatible because if the data center were built with traditional spanning-tree technologies, it would incur two problems:

- Spanning-tree scalability limits on the control plane
- Exhaustion of the MAC address tables

To address these requirements, the ACI fabric is built based on a VXLAN overlay, which allows switches to maintain perceived Layer 2 adjacency on top of a Layer 3 network, thus removing the control plane load associated with spanning tree from the switching infrastructure. To address the mobility requirements over a Layer 3 infrastructure, the forwarding is based on host-based forwarding of full /32 addresses combined with the mapping database.

This overlay, like most, requires the data path at the edge of the network to map from the tenant end point address in the packet, a.k.a. its *identifier*, to the location of the endpoint, a.k.a. its *locator*. This mapping occurs in a function called a *tunnel endpoint* (TEP). The challenge with this mapping is having to scale for very large data centers, because the mapping state must exist in many network devices.

The second problem with scale is that when an endpoint moves (that is, its locator changes), the mapping state must be updated across the network in all TEPs that have that mapping.

The ACI solution addresses these problems by using a combination of a centralized database of the mappings implemented in the packet data path, at line rate, and a caching mechanism, again in the data path, at the TEP. (Chapter 7, “ACI Fabric Design Methodology,” explains the traffic forwarding in ACI in detail.)

The other key requirement of building a cloud solution is to be able to instantiate networks in a programmatic way. If the network is managed box by box, link by link, the script or the automation tool must access individual boxes and trace where a workload is in order to enable VLAN trunking on a number of links. It must also ensure that the end-to-end path is provisioned according to the abstraction model. ACI solves this issue by providing a centralized configuration point, the APIC controller, while still maintaining individual control plane capabilities on each node in the fabric. The controller exposes the entire network as a hierarchy of objects in a tree. It describes network properties related to workloads as logical properties instead of physical properties. So, to define connectivity requirements for workloads, you don't have to express which physical interface a particular workload is on.

Furthermore, the fabric exposes the networking properties of all the switches so that they can all be configured and managed via Representational State Transfer (REST) calls as a single giant switch/router. The APIC REST API accepts and returns HTTP or HTTPS

messages that contain JSON or XML documents. Orchestration tools can easily program the network infrastructure by using REST calls. (Chapter 4, “Operational Model,” illustrates this new model and how to automate configurations with REST calls and scripting.)

Multi-tenancy is conveyed in the management information model by expressing all configurations of bridge domains, VRF contexts, and application network profile as children of an object of type `fvTenant`. The segmentation on the network transport is guaranteed by the use of different VXLAN VNIDs.

Insertion of firewall and load balancers is also automated to simplify the creation of virtual containers comprising physical or virtual firewall and load balancing services. (Chapter 8, “Service Insertion with ACI,” illustrates in more detail the modeling of services and how they are added to the fabric.)

Amazon Web Services Model

This section describes some of the services offered by Amazon Web Services and some of the AWS naming conventions. AWS offers a very wide variety of services, and the purpose of this section is not to describe all of them. Rather, this section is useful to the network administrator for two reasons:

- As a reference for a popular IaaS service
- The potential need to extend a private cloud into the Amazon Virtual Private Cloud

The following list provides some key AWS terminology:

- **Availability Zone:** A distinct location within a region that is insulated from failures in other Availability Zones, and provides inexpensive, low-latency network connectivity to other Availability Zones in the same region.
- **Region:** A collection of Availability Zones, such as `us-west`, `us-east-1a`, `eu-west`, etc., in the same geographical region
- **Access credentials:** A public key that is used to access AWS resources allocated to a given user
- **Amazon Machine Image (AMI):** The image of a given virtual machine (which Amazon calls an *instance*)
- **Instance:** A virtual machine that is running a given AMI image
- **Elastic IP address:** A static address associated with an instance

Amazon Elastic Compute Cloud (EC2) services enable you to launch an AMI in a region of the user’s choice and in an Availability Zone of the user’s choice. Instances are protected by a firewall. The instance also gets an IP address and a DNS entry. The EC2 services can also be accompanied by the Elastic Load Balancing, which distributes

traffic across EC2 compute instances. Auto Scaling helps with provisioning enough EC2 instances based on the utilization. Amazon CloudWatch provides information about CPU load, disk I/O rate, and network I/O rate of each EC2 instance.

Note More information can be found at:

<http://docs.aws.amazon.com/general/latest/gr/glos-chap.html>

http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/Using_Query_API.html

Amazon Simple Storage Service (S3) is accessed via web services API based on SOAP or with the HTTP API that uses the standard HTTP verbs (GET, PUT, HEAD, and DELETE). The objects are identified by using the protocol name, the S3 endpoint (s3.amazonaws.com), the object key, and what is called the *bucket name*.

All resources can be created and manipulated by using Amazon SDKs available for various programming languages, such as the Python and PHP SDKs available at the following respective URLs:

<http://aws.amazon.com/sdk-for-python/>

<http://aws.amazon.com/sdk-for-php/>

With this approach, you can fully automate tasks such as the following:

- Locating the server resources
- Attaching storage
- Providing Internet connectivity
- Setting up switching and routing
- Booting the server
- Installing the OS
- Configuring applications
- Assigning IP addresses
- Configuring firewalling
- Scaling up the infrastructure

Note For more information, please refer to the book *Host Your Web Site in the Cloud: Amazon Web Services Made Easy*, by Jeff Barr (SitePoint, 2010).

You can access the AWS-hosted Amazon Virtual Private Cloud (VPC) in multiple ways. One way is to set a jumphost to which you log in over SSH with the public key that

AWS generates. Another approach is to connect the enterprise network to the Amazon VPC via VPNs.

Automating Server Provisioning

In large-scale cloud deployments with thousands of physical and virtual servers, administrators must be able to provision servers in a consistent and timely manner.

This section is of interest to the network administrator for several reasons:

- Some of these technologies can also be used to maintain network equipment designs.
- Cisco ACI reuses some of the concepts from these technologies that have proven to be effective to the task of maintaining network configurations.
- A complete design of ACI must include support for these technologies because the compute attached to ACI will use them.

The high-level approach to automating server provisioning consists of performing the following:

- PXE booting a server (physical or virtual)
- Deploying the OS or customized OS on the server with Puppet/Chef/CFEngine agents

Because of the above reasons, a typical setup for a cloud deployment requires the following components:

- A DHCP server
- A TFTP server
- An NFS/HTTP or FTP server to deliver the kickstart files
- A master for Puppet or Chef or similar tools

PXE Booting

In modern data centers, administrators rarely install new software via removable media such as DVDs. Instead, administrators rely on PXE (Preboot eXecution Environment) booting to image servers.

The booting process occurs in the following sequence:

1. The host boots up and sends a DHCP request.
2. The DHCP server provides the IP address and the location of the PXE/TFTP server.
3. The host sends a TFTP request for pxelinux.0 to the TFTP server.

4. The TFTP server provides pxelinux.0.
5. The host runs the PXE code and requests the kernel (vmlinuz).
6. The TFTP server provides vmlinuz code and provides the location of the kickstart configuration files (NFS/HTTP/FTP and so on).
7. The host requests the kickstart configuration from the server.
8. The HTTP/NFS/FTP server provides the kickstart configuration.
9. The host requests to install packages such as the RPMs.
10. The HTTP/NFS/FTP server provides the RPMs.
11. The host runs Anaconda, which is the post-installation scripts.
12. The HTTP/NFS/FTP server provides the scripts and the Puppet/Chef installation information.

Deploying the OS with Chef, Puppet, CFEngine, or Similar Tools

One of the important tasks that administrators have to deal with in large-scale data centers is maintaining up-to-date compute nodes with the necessary level of patches, the latest packages, and with the intended services enabled.

You can maintain configurations by creating VM templates or a golden image and instantiating many of them, but this process produces a monolithic image, and replicating this process every time a change is required is a lengthy task. It is also difficult, if not impossible, to propagate updates to the configuration or libraries to all the servers generated from the template. The better approach consists of using a tool such as Chef, Puppet, or CFEngine. With these tools, you create a bare-bones golden image or VM template and you push servers day-2.

These tools offer the capability to define the node end state with a language that is abstracted from the underlying OS. For instance, you don't need to know whether to install a package with "yum" or "apt"; simply define that a given package is needed. You don't have to use different commands on different machines to set up users, packages, services, and so on.

If you need to create a web server configuration, define it with a high-level language. Then, the tool creates the necessary directories, installs the required packages, and starts the processes listening on the ports specified by the end user.

Some of the key characteristics of these tools are that they are based on principles such as a "declarative" model (in that they define the desired end state) and idempotent configurations (in that you can rerun the same configuration multiple times and it always yields the same result). The policy model relies on the declarative approach. (You can find more details about the declarative model in Chapter 3, "The Policy Data Center.")

With these automation tools, you can also simulate the result of a given operation before it is actually executed, implement the change, and prevent configuration drifting.

Chef

The following list provides a reference for some key terminology used by Chef:

- **Node:** The server (but could be a network device).
- **Attributes:** The configuration of a node.
- **Resources:** Packages, services, files, users, software, networks, and routes.
- **Recipe:** The intended end state of a collection of resources. It is defined in Ruby.
- **Cookbook:** The collection of recipes, files, and so on for a particular configuration need. A cookbook is based on a particular application deployment and defines all the components necessary for that application deployment.
- **Templates:** Configuration files or fragments with embedded Ruby code (.erb) that is resolved at run time.
- **Run list:** The list of recipes that a particular node should run.
- **Knife:** The command line for Chef.
- **Chef client:** The agent that runs on a node.

Normally the administrator performs configurations from “Knife” from a Chef workstation, which has a local repository of the configurations. The cookbooks are saved on the Chef server, which pushes them to the nodes, as shown in Figure 2-2.

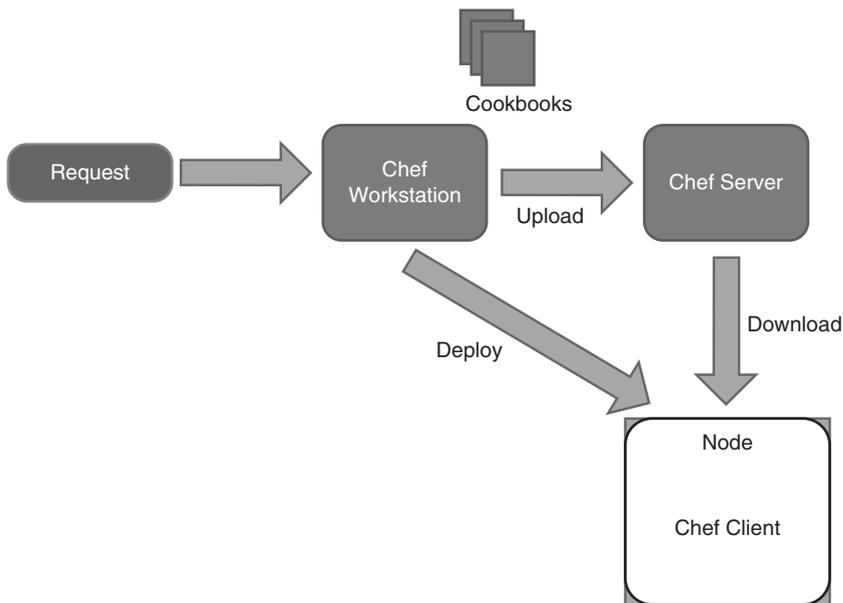


Figure 2-2 *Chef Process and Interactions*

The recipe that is relevant to the action to be performed on the device is configured on the Chef workstation and uploaded to the Chef server.

Puppet

Figure 2-3 illustrates how Puppet operates. With the Puppet language, you define the desired state of resources (users, packages, services, and so on), simulate the deployment of the desired end state as defined in the manifest file, and then apply the manifest file to the infrastructure. Finally, it is possible to track the components deployed, track the changes, and correct configurations from drifting from the intended state.

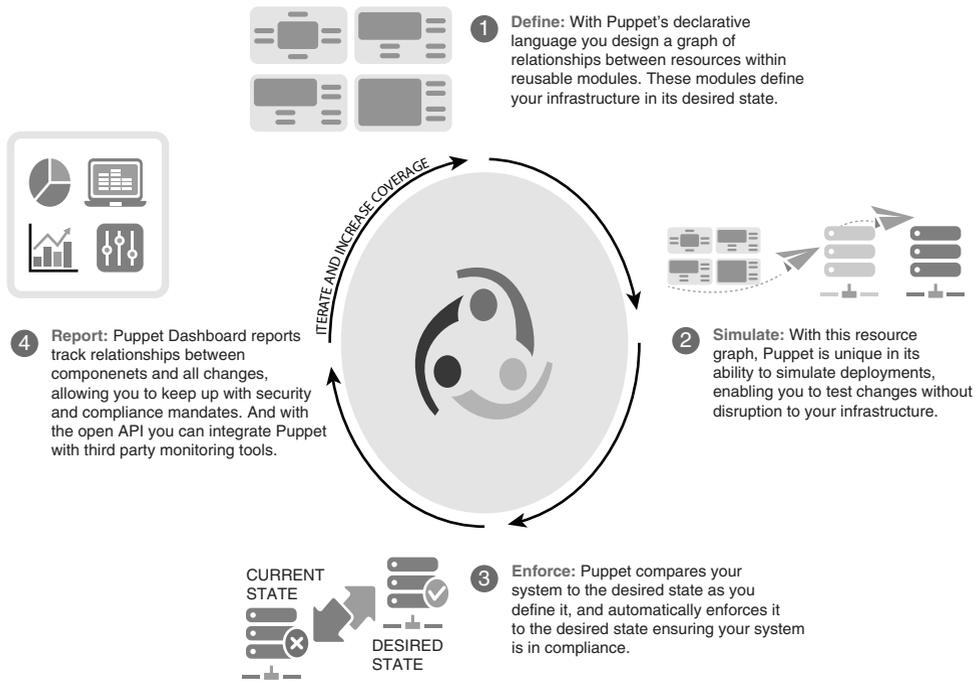


Figure 2-3 *Puppet*

The following is a list of some key terminology used in Puppet:

- **Nodes:** The servers, or network devices
- **Resource:** The object of configuration: packages, files, users, groups, services, and custom server configuration.
- **Manifest:** A source file written using Puppet language (.pp)
- **Class:** A named block of Puppet code
- **Module:** A collection of classes, resource types, files, and templates, organized around a particular purpose

- **Catalog:** Compiled collection of all resources to be applied to a specific node, including relationships between those resources

Orchestrators for Infrastructure as a Service

Amazon EC2, VMware vCloud Director, OpenStack, and Cisco UCS Director are IaaS orchestrators that unify the provisioning of virtual machines, physical machines, storage, and networking and can power up the entire infrastructure for a given user environment (called a *container*, *virtual data center*, or *tenant*).

The following common operations are enabled by these tools:

- Creating a VM
- Powering up a VM
- Powering down a VM
- Power cycling a VM
- Changing ownership of a server
- Taking a snapshot of an image

vCloud Director

VMware supports the implementation of clouds with the use of vCloud Director. vCloud Director builds on top of vSphere, which in turn coordinates VMs across a number of hosts that are running vSphere. Figure 2-4 illustrates the features of vCloud Director, which provides tenant abstraction and resource abstraction and a vApp Catalog for users of the cloud computing service.

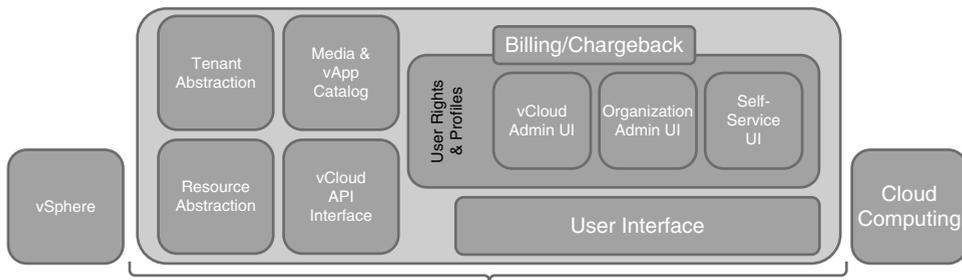


Figure 2-4 vCloud Director Components

Figure 2-5 shows how vCloud Director organizes resources in a different way and provides them as part of a hierarchy where the Organization is at the top. Inside the Organization there are multiple vDCs.

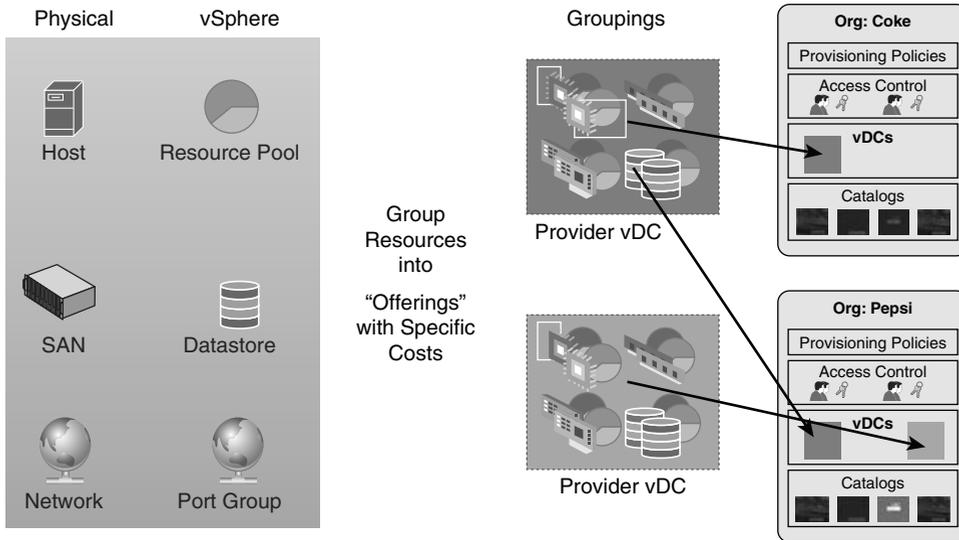


Figure 2-5 *vCloud Director Organization of Resources*

OpenStack

Chapter 6, “OpenStack,” covers the details of OpenStack as it relates to ACI. The purpose of this section is to explain how OpenStack fits in cloud architectures.

Project and Releases

Each functional area of OpenStack is a separate project. For the purpose of cloud deployments, you don’t have to use the entire OpenStack set of capabilities; you can, for instance, just leverage the APIs of a particular project.

The list of projects is as follows:

- Nova for compute
- Glance, Swift, and Cinder for image management, object storage, and block storage, respectively
- Horizon for the dashboard, self-service portal, and GUI
- Neutron for networking and IP address management
- Telemetry for metering
- Heat for orchestration

The release naming is very important because different releases may have significant changes in capabilities. At the time of this writing, you may encounter the following releases:

- Folsom (September 27, 2012)
- Grizzly (April 4, 2013)
- Havana (October 17, 2013)
- Icehouse (April 17, 2014)
- Juno (October 2014)
- Kilo (April 2015)

Note You can find the list of releases at:

<http://docs.openstack.org/training-guides/content/associate-getting-started.html#associate-core-projects>

The releases of particular interest currently for the network administrator are Folsom, because it introduced the Quantum component to manage networking, and Havana, which replaced the Quantum component with Neutron. Neutron gives more flexibility to manage multiple network components simultaneously, especially with the ML2 architecture, and is explained in detail in Chapter 6.

The concept of the plug-in for Neutron is significant. It is how networking vendors plug into the OpenStack architecture. Neutron provides a plug-in that can be used by OpenStack to configure their specific networking devices through a common API.

Multi-Hypervisor Support

OpenStack manages compute via the Nova component, which controls a variety of compute instances, such as the following:

- Kernel-based Virtual Machine (KVM)
- Linux Containers (LXC), through libvirt
- Quick EMUlator (QEMU)
- User Mode Linux (UML)
- VMware vSphere 4.1 update 1 and newer
- Xen, Citrix XenServer, and Xen Cloud Platform (XCP)
- Hyper-V
- Baremetal, which provisions physical hardware via pluggable subdrivers

Installers

The installation of OpenStack is a big topic because installing OpenStack has been complicated historically. In fact, Cisco took the initiative to provide an OpenStack rapid scripted installation to facilitate the adoption of OpenStack. At this time many other installers exist.

When installing OpenStack for proof-of-concept purposes, you often hear the following terminology:

- **All-in-one installation:** Places the OpenStack controller and nodes' components all on the same machine
- **Two-roles installation:** Places the OpenStack controller on one machine and a compute on another machine

To get started with OpenStack, you typically download a devstack distribution that provides an all-in-one, latest-and-greatest version. Devstack is a means for developers to quickly “stack” and “unstack” an OpenStack full environment, which allows them to develop and test their code. The scale of devstack is limited, naturally.

If you want to perform an all-in-one installation of a particular release, you may use the Cisco installer for Havana by following the instructions at <http://docwiki.cisco.com/wiki/OpenStack:Havana:All-in-One>, which use the git repo with the code at https://github.com/CiscoSystems/puppet_openstack_builder. Chapter 6 provides additional information regarding the install process.

There are several rapid installers currently available, such as these:

- Red Hat OpenStack provides PackStack and Foreman
- Canonical/Ubuntu provides Metal as a Service (MaaS) and Juju
- SUSE provides SUSE Cloud
- Mirantis provides Fuel
- Piston Cloud provides one

Architecture Models

When deploying OpenStack in a data center, you need to consider the following components:

- A PXE server/Cobbler server (Quoting from Fedora: “Cobbler is a Linux installation server that allows for rapid setup of network installation environments. It glues together and automates many associated Linux tasks so you do not have to hop between lots of various commands and applications when rolling out new systems, and, in some cases, changing existing ones.”)
- A Puppet server to provide image management for the compute nodes and potentially to image the very controller node of OpenStack
- A node or more for OpenStack controllers running keystone, Nova (api, cert, common, conductor, scheduler, and console), Glance, Cinder, Dashboard, and Quantum with Open vSwitch

- The nodes running the virtual machines with Nova (common and compute) and Quantum with Open vSwitch
- The nodes providing the proxy to the storage infrastructure

Networking Considerations

Cisco products provide plug-ins for the provisioning of network functionalities to be part of the OpenStack orchestration. Figure 2-6 illustrates the architecture of the networking infrastructure in OpenStack.

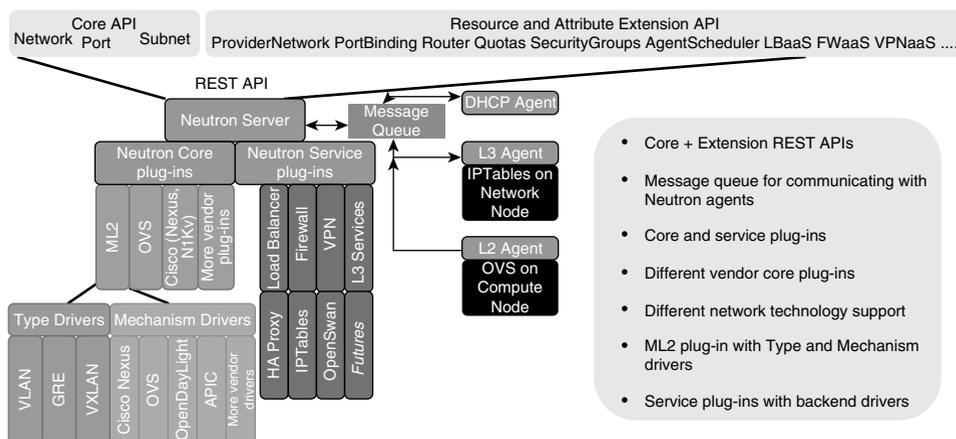


Figure 2-6 *OpenStack Networking Plug-ins*

Networks in OpenStack represent an isolated Layer 2 segment, analogous to VLAN in the physical networking world. They can be mapped to VLANs or VXLANs and become part of the ACI End Point Groups (EPGs) and Application Network Policies (ANP). As Figure 2-6 illustrates, the core plug-ins infrastructure offers the option to have vendor plug-ins. This topic is described in Chapter 6.

Note For more information about OpenStack, visit <http://www.openstack.org>.

UCS Director

UCS Director is an automation tool that allows you to abstract the provisioning from the use of the element managers and configure compute, storage, and ACI networking as part of an automated workflow in order to provision applications. The workflow provided by UCS Director is such that the administrator defines server policies, application network policies, storage policies, and virtualization policies, and UCSD applies these policies across the data center as shown in Figure 2-7.

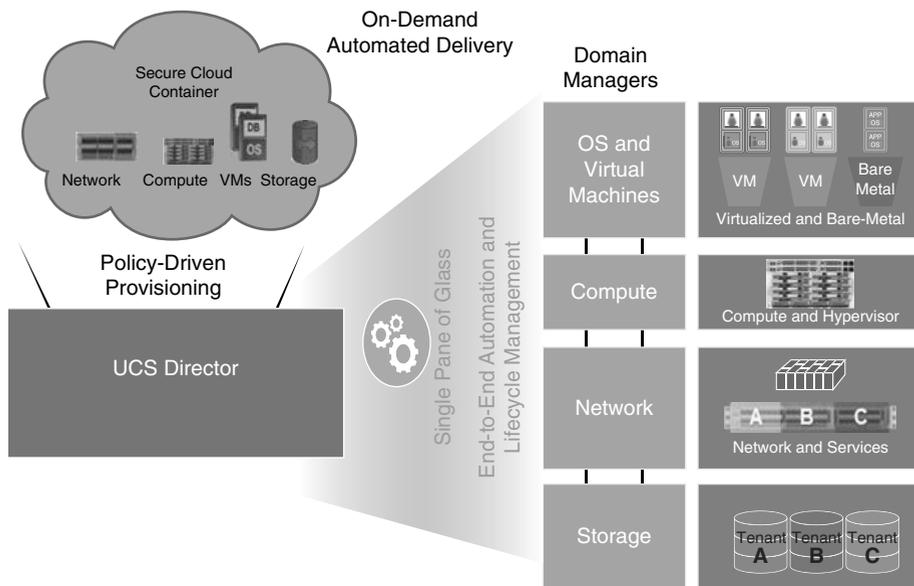


Figure 2-7 UCS Director

The workflow can be defined in a very intuitive way via the graphical workflow designer.

UCSD has both a northbound API and a southbound API. The southbound API allows UCSD to be an extensible platform.

Note For additional information on UCS Director, visit: <https://developer.cisco.com/site/data-center/converged-infrastructure/ucs-director/overview/>

Cisco Intelligent Automation for Cloud

Cisco Intelligent Automation for Cloud is a tool that enables a self-service portal and is powered by an orchestration engine to automate the provisioning of virtual and physical servers. Although there are some blurred lines between UCSD and CIAC, CIAC uses the UCSD northbound interface and complements the orchestration with the ability to standardize operations such as offering a self-service portal, opening a ticket, doing charge-back, and so on. CIAC orchestrates across UCSD, OpenStack, and Amazon EC2, and integrates with Puppet/Chef. It also provides measurement of the utilization of resources for the purpose of pricing. Resources being monitored include vNIC, hard drive usage, and so on.

Figure 2-8 illustrates the operations performed by CIAC for PaaS via the use of Puppet.

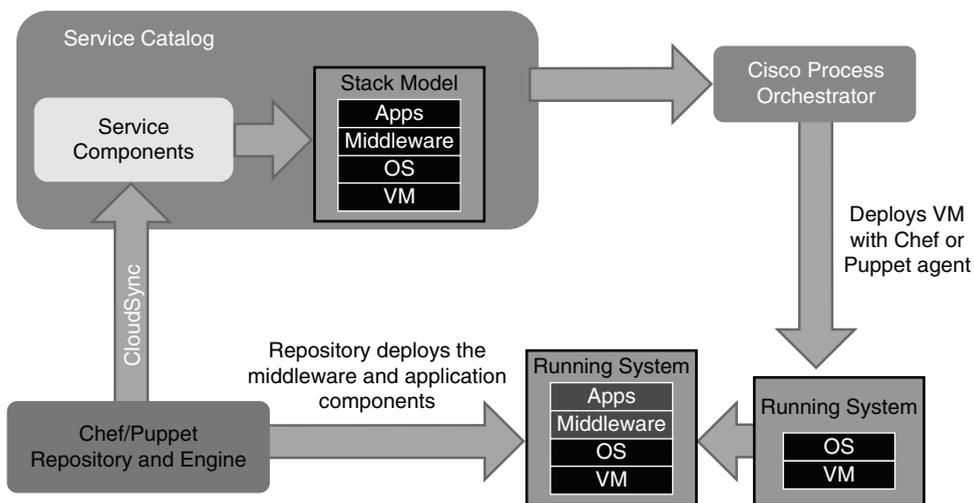


Figure 2-8 CIAC Operations

Figure 2-9 illustrates more details of the provisioning part of the process.

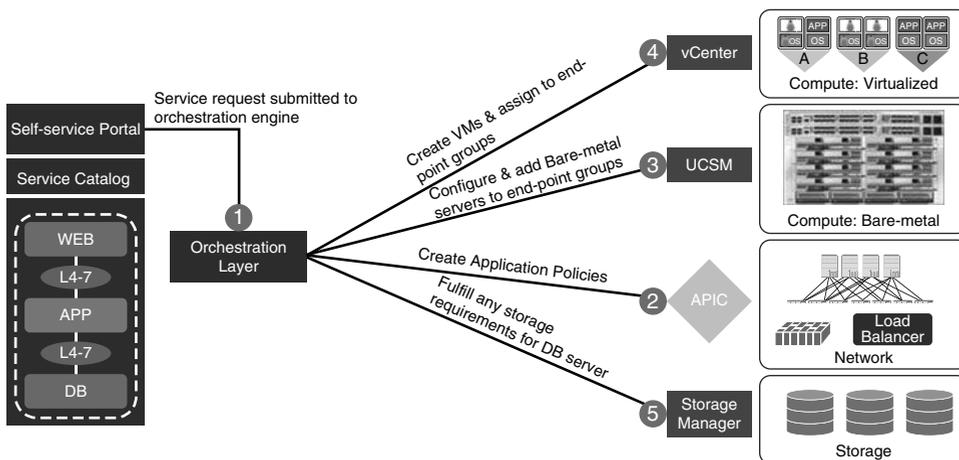


Figure 2-9 CIAC Workflow

CIAC organizes the data center resources with the following hierarchy:

- Tenants
- Organization within tenants
- Virtual data centers
- Resources

Figure 2-10 illustrates the hierarchy used by CIAC.

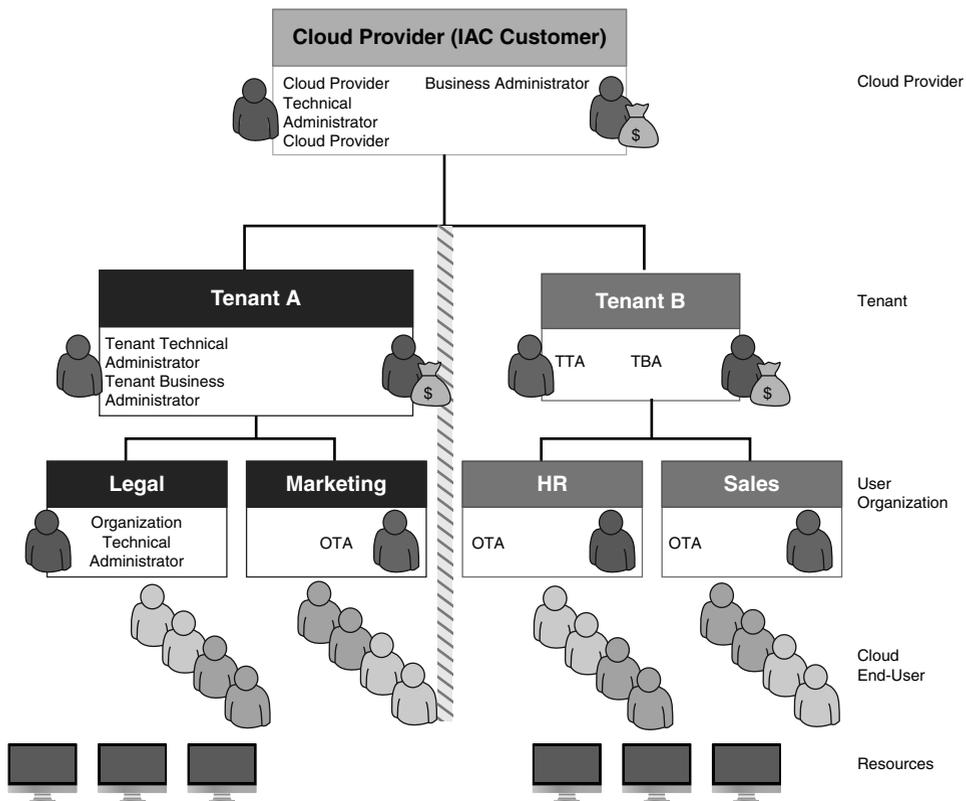


Figure 2-10 *Hierarchy in CIAC*

The user is offered a complete self-service catalog that includes different options with the classic Bronze, Silver, and Gold “containers” or data centers to choose from, as illustrated in Figure 2-11.

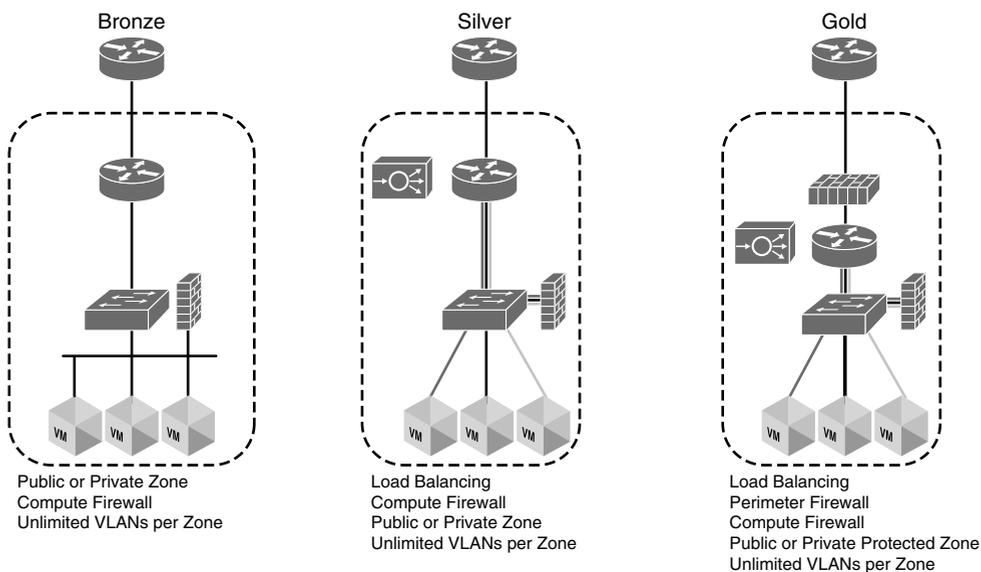


Figure 2-11 Containers

Conciliating Different Abstraction Models

One of the tasks of an administrator is to create a cloud infrastructure that maps the abstraction model of the service being offered to the abstractions of the components that make the cloud.

A typical offering may consist of a mix of VMware-based workloads, OpenStack/KVM-based workloads with an ACI network, and UCS/CIAC orchestration. Each technology has its own way of creating hierarchy and virtualizing the compute and network.

Table 2-1 provides a comparison between the different environments.

Table 2-1 Differences Among VMware vCenter Server, VMware vCloud Director, OpenStack, Amazon EC2, UCS Director, CIAC, and ACI

Platform Type/Property	VMware vCenter Server	VMware vCloud Director	OpenStack (Essex)	Amazon AWS (EC2)	UCS Director	CIAC	ACI
Compute POD	Data center	Organization	OpenStack PE ID	Account	Account	Server	N/A
Tenant	Folder	Organization	N/A	Account	N/A	Tenant	Security domain
Organization	Folder	N/A	N/A	N/A	Group	Organization	Tenant
VDC	Resource pool	Organization VDC	Project	Account	VDC	VDC	Tenant

Platform Type/Property	VMware vCenter Server	VMware vCloud Director	OpenStack (Essex)	Amazon AWS (EC2)	UCS Director	CIAC	ACI
VLAN Instance	vCenter network	Org network/network pool	Network ID	Network ID	Network policy	Network	Subnet
VM Template	Full path	VM template HREF	Image ID	AMI ID	Catalog	Server template	N/A

In ACI the network is divided into tenants, and the administration of the tenants is organized with the concept of a security domain. Different administrators are associated with one or more security domains and, similarly, each tenant network can be associated with one or more security domains. The result is a many-to-many mapping, which allows creating sophisticated hierarchies. Furthermore, if two tenant networks represent the same “tenant” in CIAC but two different organizations within the same “tenant,” it is possible to share resources and enable the communication between them.

In CIAC, a tenant can contain different organizations (e.g., departments) and each organization can own one or more virtual data centers (aggregates of physical and virtual resources). Network and other resources can be either shared or segregated, and the API exposed by the ACI controller (APIC) to the orchestrator makes it very easy.

Note For more information regarding Cisco’s development in the OpenStack area, visit these links:

<http://www.cisco.com/web/solutions/openstack>

<http://docwiki.cisco.com/wiki/OpenStack>

Summary

This chapter described the components of a cloud infrastructure and how ACI provides network automation for the cloud. It explained the Amazon Web Services approach. This chapter also described the role of the various orchestration tools, such as OpenStack, Cisco UCS Director, and Cisco Intelligent Automation for Cloud. It also introduced some key concepts regarding how to automate the provisioning of servers and how to get started with OpenStack. It explained the OpenStack modeling of the cloud infrastructure and compared it to similar modeling by CIAC and ACI. It also discussed the administrator’s task of mapping the requirements of IaaS services onto the models of these technologies.

Index

Numbers

10-Gigabit Ethernet cabling, 208

40-Gigabit Ethernet cabling, 208

A

AaaS (Application as a Service), 38

abstract nodes

connecting, 258-260

connector definition, 257-258

elements, 258

function definition, 255-257

access control for APIC policy model, 88-89

access credentials, 41

access ports, configuring, 223-228

ACI (Application Centric Infrastructure). *See also* APIC (Application Policy Infrastructure Controller) model

in cloud computing, network requirements, 39-41

design, 59-61

benefits, 193

configuring, 219-235

forwarding, 194-202

hardware and software requirements, 207-208

management model, 204-207

multitenancy, 218-219

physical topology, 208-218

segmentation with endpoint groups, 202-204

virtual topology configuration, 235-241

goals of, 248-249

health monitoring, 272-281

events and logs, 279-280

faults, 274-279

health score, 280-281

statistics, 273-274

OpenStack and

benefits, 177-180

features and capabilities, 191

integration, 180-181

- operational model, 91
 - additional resources*, 124
 - APIC interfaces*, 106-108
 - Git/GitHub*, 103-106
 - JSON*, 94-95
 - network management options*, 92-93
 - object tree*, 108-114
 - Python*, 96-103
 - Python SDK for ACI*, 122-124
 - REST calls*, 114-122
 - REST protocol*, 93-94
 - XML*, 94
 - YAML*, 95-96
- service insertion, 243
 - benefits*, 244
 - concrete and logical devices*, 250-251
 - configuring*, 252-266
 - connecting EPGs with service graph*, 244-245
 - defining service graph*, 249-250
 - hardware and software requirements*, 247-248
 - logical device selectors*, 251
 - management model*, 245
 - rendering service graph*, 246-249
 - splitting bridge domains*, 251
 - for virtualized servers*, 245
- telemetry. *See* telemetry
- virtual server connectivity, 160-165
 - endpoint discovery*, 162
 - Hyper-V integration*, 162-163
 - KVM integration*, 163-164
 - overlay normalization*, 160-161
 - policy resolution immediacy*, 162
 - VMM domain*, 161-162
 - VMware ESX/ESXi integration*, 164-165
- virtualized data centers, advantages of, 128
- ACI Fabric OS, 79-80
- actions (APIC model), 75
- AEP (attach entity profile), 217-218, 234-235
- Amazon Elastic Compute Cloud (EC2) services, 41-42
- Amazon Machine Image (AMI), 41
- Amazon Simple Storage Service (S3), 42
- Amazon Virtual Private Cloud (VPC), 43
- Amazon Web Services (AWS) model, 41-43
- AMI (Amazon Machine Image), 41
- ANPs (application network profiles), 70-71, 118
- any-to-any policy, 121-122
- APIC (Application Policy Infrastructure Controller) model, 57, 207
 - access control, 88-89
 - ACI Fabric OS, 79-80
 - ANPs (application network profiles), 70-71
 - atomic counters, 270
 - benefits, 178
 - component architecture, 80
 - Appliance Director*, 83
 - Appliance Element*, 84

- Boot Director*, 82-83
- Event Manager*, 83
- Observer*, 82
- Policy Manager*, 81
- Topology Manager*, 81-82
- VMM Manager*, 83
- contracts, 71-79
- design considerations, 210-211
- EPGs (endpoint groups), 62-65
- logical object model, 61-63
- OpenStack driver, 181
 - configuring*, 185-188
 - features and capabilities*, 191
 - installing*, 183-185
- OpenStack integration, 180-181
- policy enforcement
 - multicast*, 69-70
 - unicast*, 66-68
- promise theory, 59-61
- sharding, 84-87
- subjects, 73
- taboos, 74-75
- user interface
 - CLI (command-line interface)*, 87
 - GUI (graphical user interface)*, 87
 - RESTful API*, 88
 - strengths and weaknesses*, 106-108
- Appliance Director**, 83
- Appliance Element**, 84
- application and storage designs (data center architectures)**, 1-2
- big data data centers, 7-13
- high-performance compute data centers (HPC), 14-15
- massively scalable data centers (MSDC), 21-25
- ultra-low latency data centers (ULL), 16-20
- virtualized data centers, 2-7
- Application as a Service (AaaS)**, 38
- Application Centric Infrastructure**. *See* **ACI (Application Centric Infrastructure)**
- application deployment in Policy Driven Data Center model**, need for, 57-59
- application network profiles (ANPs)**, 70-71, 118
- Application Policy Infrastructure Controller (APIC)**. *See* **APIC (Application Policy Infrastructure Controller) model**
- atomic counters**, 267-270
- attach entity profile (AEP)**, 217-218, 234-235
- attacks, preventing with CoPP (Control Plane Policing)**, 288-291
- audit logs**, 280
- authentication for APIC policy model**, 88-89
- authorization for APIC policy model**, 88-89
- automating**
 - policies,
 - server provisioning, 43
 - OS deployment*, 44-47
 - PXE booting*, 43-44
- availability, big data data centers**, 12
- Availability Zones**, 41
- AWS (Amazon Web Services) model**, 41-43

B

- Bare Metal Service, 174-175**
- big data data centers, 7-13**
 - cluster design, 10
 - components, 8
 - design requirements, 11-13
 - availability and resiliency, 12*
 - burst handling and queue depth, 12-13*
 - data node network speed, 13*
 - network latency, 13*
 - oversubscription ratio, 13*
 - in enterprise data models, 8
 - network requirements, 9
 - QoS (quality of service), 312
 - storage requirements, 11
- blacklist model (taboos), 74-75**
- Boot Director, 82-83**
- boot process, PXE booting, 43-44**
- brctl command, 142-143**
- bridge domain, 237-238, 251**
- bridging, Linux KVM, 142-143**
- broadcast traffic, forwarding, 213-214**
- buffer bloat, 317-318**
- buffer strategies**
 - big data data centers, 12-13
 - QoS (quality of service), 315-317
- bundles (APIC model), 78**
- burst handling, big data data centers, 12-13**

C

- centralized repositories, 104**
- centralized shared memory (SoC), 306-309**
- Chef, 45-46**
- CIAC (Cisco Intelligent Automation for Cloud), 52-54**
- Cinder, 173**
- Cisco ACI. *See* ACI (Application Centric Infrastructure)**
- Cisco APIC. *See* APIC (Application Policy Infrastructure Controller) model**
- Cisco Intelligent Automation for Cloud (CIAC), 52-54**
- Cisco Nexus switches**
 - flowlet switching, 322-323
 - Nexus 1000V, 155-158
 - Nexus 9000 series, 208
 - QoS (quality of service) implementation, 324-326
 - switch architectures by model, 326
- classes (ACI), 109-113**
- classification boundaries, 313-314**
- CLI (command-line interface)**
 - for APIC policy model, 87
 - MQC (Modular QoS CLI) model, 324-326
- cloud computing**
 - Amazon Web Services (AWS) model, 41-43
 - automating server provisioning, 43
 - OS deployment, 44-47*
 - PXE booting, 43-44*
 - explained, 37-39

- IaaS orchestrators, 47-56
 - Cisco Intelligent Automation for Cloud (CIAC)*, 52-54
 - comparison of models*, 55-56
 - OpenStack*, 48-51. See also *OpenStack*
 - UCS Director*, 51-52
 - vCloud Director*, 47
- network requirements, 39-41
- POD model, 26-28
- cloud delivery model, 37
- cloud orchestration, 128
- cluster design, big data data centers, 10
- code listings. *See* examples
- CodeTalker, 101
- collision model, 301
- command-line interface (CLI)
 - for APIC policy model, 87
 - MQC (Modular QoS CLI) model, 324-326
- commands, Git/GitHub, 105-106
- concrete devices, 250-251, 260-266
- configuring
 - APIC driver, 185-188
 - Cisco ACI fabric, 219-235
 - interface policy groups*, 229
 - network management*, 221-223
 - policy-based configuration of access ports*, 223-228
 - PortChannels*, 229-231
 - requirements*, 219-220
 - virtual PortChannels (vPCs)*, 231-232
 - VMM domains*, 233-235
 - zero-touch provisioning*, 220-221
- interface policies, 228
- logical device selectors, 264-265
- policy groups, 229
- PortChannel, 188, 229-231
- service insertion, 252-266
 - abstract node connectors*, 257-258
 - abstract node elements*, 258
 - abstract node functions*, 255-257
 - concrete and logical device definition*, 260-266
 - connecting abstract nodes*, 258-260
 - metadevices*, 254-255
 - naming conventions*, 265-266
 - service graph boundaries*, 253
- switch profiles, 228
- virtual topology, 235-241
 - bridge domain*, 237-238
 - endpoint connectivity*, 238-240
 - external connectivity*, 240-241
- vPCs (virtual PortChannels), 231-232
- contexts (APIC model), 62-63
- contracts (APIC model), 71-79, 120-121, 244-245
- Control Plane Policing (CoPP), 288-291
- control planes
 - CoPP (Control Plane Policing), 288-291
 - interaction between data and management planes, 287-288
 - packet classification, 290
 - packet types, 288-290
 - rate-controlling mechanisms, 290-291
 - separation between data and management planes, 286

control traffic, 155

Controlling Bridge, 158-160

CoPP (Control Plane Policing), 288-291

credit model, 301

crossbar switch fabric architecture, 295-306

- benefits, 297
- cut-through switching, 301-302
- HOLB (head-of-line blocking), 304
- input queuing, 303-304
- multicast switching, 298
- multistage crossbar fabrics, 305-306
- output queuing, 302-303
- overspeed, 298
- scheduler, 301
- with SoC, 306-309
- superframing, 299-301
- unicast switching, 297
- VoQ (virtual output queuing), 304

cURL, 117

cut-through switching, 292-295, 301-302

D

data center architectures

- application and storage designs, 1-2
 - big data data centers*, 7-13
 - high-performance compute data centers (HPC)*, 14-15
 - massively scalable data centers (MSDC)*, 21-25
 - ultra-low latency data centers (ULL)*, 16-20
 - virtualized data centers*, 2-7

- cloud computing. *See* cloud computing
- designs
 - end of row (EoR) model*, 29-30
 - middle of row (MoR) model*, 30
 - project requirements*, 29
 - spine-leaf model*, 33-35
 - top of rack (ToR) model*, 30-32
- FlexPod model, 28
- POD model, 26-28
- Policy Driven Data Center
 - need for*, 57-59
- switch architecture
 - centralized shared memory (SoC)*, 306-309
 - by Cisco Nexus model*, 326
 - CoPP (Control Plane Policing)*, 288-291
 - crossbar switch fabric architecture*, 295-306
 - cut-through switching*, 292-295
 - data, control, management plane interaction*, 287-288
 - data, control, management plane separation*, 286
 - requirements*, 291
 - summary of*, 291
- Data Center Bridging Exchange (DCBX), 320
- Data Center TCP (DCTCP), 320-321
- data node network speed, big data data centers, 13
- data planes, 286-288
- data structures, Python, 98-99
- DCBX (Data Center Bridging Exchange), 320
- DCTCP (Data Center TCP), 320-321

declarative management model in
 ACI fabric, 204-207

defining

- abstract node connectors, 257-258
- abstract node functions, 255-257
- concrete devices, 260-266
- logical devices, 260-266
- service graph, 249-250
- service graph boundaries, 253

denial of service (DoS) attacks,
 289-290

dEPG (destination EPGs), 66

deployment, OpenStack, 176-177

- configuring APIC driver, 185-188
- example, 182-189
- installing Icehouse, 183-185
- troubleshooting, 188-189

design requirements

- big data data centers, 11-13
 - availability and resiliency, 12*
 - burst handling and queue depth, 12-13*
 - data node network speed, 13*
 - network latency, 13*
 - oversubscription ratio, 13*
- high-performance compute data centers (HPC), 14-15
- massively scalable data centers (MSDC), 24
- ultra-low latency data centers (ULL), 18-19
- virtualized data centers, 6

design topologies

- high-performance compute data centers (HPC), 15
- massively scalable data centers (MSDC), 25

summary of, 25

ultra-low latency data centers (ULL),
 19-20

designs

ACI (Application Centric Infrastructure), 59-61. *See also* fabric design (ACI)

data center architectures

- end of row (EoR) model, 29-30*
- middle of row (MoR) model, 30*
- project requirements, 29*
- spine-leaf model, 33-35*
- top of rack (ToR) model, 30-32*

FlexPod model, 28

destination EPGs (dEPG), 66

dictionaries (Python), 98

distributed repositories, 104

distributed switching, 133

Distributed Virtual Switch, 149-151

domains, 216-217

- bridge domain, 237-238, 251
- VMM domains, 233-235

DoS (denial of service) attacks,
 289-290

dvPort group, 149

E

easy_install, 101

EC2 (Amazon Elastic Compute Cloud) services, 41-42

ECN (Early Congestion Notification), 320-321

eggs (Python). *See* packages (Python)

egress leaf, policy enforcement, 68

elastic IP addresses, 41

end of row (EoR) model, 29-30

- endpoint discovery, 162**
- Enhanced Transmission Selection (ETS), 319**
- enterprise data models, big data data centers in, 8**
- EoR (end of row) model, 29-30**
- EPGs (endpoint groups), 62-65, 160**
 - adding, 118
 - any-to-any policy, 121-122
 - connecting with service graph, 244-245
 - contracts, 71-79, 120-121
 - endpoint connectivity, 238-240
 - segmentation with, 202-204
 - in server virtualization, 128, 133
- errors in Python scripts, 101**
- ESX/ESXi, 149-154**
 - ACI integration, 164-165
 - traffic requirements, 151-152
 - vCloud Director and vApp, 152-154
 - vSwitch and distributed virtual switches, 150-151
- Eth0 interface, 287-288**
- Eth1 interface, 288**
- Eth3 interface, 287**
- Eth4 interface, 287**
- ETS (Enhanced Transmission Selection), 319**
- event logs, 280**
- Event Manager, 83**
- events, health monitoring, 279-280**
- examples**
 - Association of Ports with the vPC Channel Group, 232
 - Configuration of a PortChannel, 230
 - Configuration of a vPC Channel Group, 232
 - Configuration of Concrete Device that Is a Physical Appliance, 263
 - Configuration of Concrete Device that Is a Virtual Appliance, 263
 - Configuration of Connectivity to a Physical Server, 239
 - Configuration of Connectivity to a Virtualized Server, 240
 - Configuration of External Connectivity, 240
 - Configuration of the vPC Protection Group, 231
 - Configuring an OVS Switch, 146
 - Connecting the Node to the Boundary of the Graph, 259
 - Connecting Two vnsAbsNodes Together, 259
 - Contract that Connects Two EPGs Has a Reference to a Graph, 254
 - Creating a Virtual Environment for Cobra, 123
 - cURL to Send REST Calls, 117
 - Defining a Function (Python), 97
 - Definition of a Contract, 121
 - Definition of a Logical Device, 262
 - Definition of an Any-to-Any Policy, 121
 - Definition of an EPG, 121
 - Definition of the Boundary of the Service Graph, 253
 - Definition of the Connectors of the Abstract Node, 257-258
 - Deleting Objects with REST Calls, 115
 - Fabric Discovery, 124
 - Importing a Module (Python), 98
 - JSON-Formatted Files, 100
 - List (Python), 98

Load-Balancing Configuration, 256

Logging In to the Fabric with the SDK, 123

Logical Device Context Base Configuration, 264-265

Mapping Virtualized Servers Mobility Domain to the Fabric, 217

ML2_cisco_conf.ini Parameters to Configure, 186

ML2_conf.ini Parameters to Configure, 186

Neutron Network API Options, 170

Neutron Subnet API Options, 170

Object NodeP in XML Format, 112

Port Profile in Cisco Nexus 1000V, 157

Python Script to Send REST Calls, 116

Querying with Cobra, 124

REST Call to Add an EPG, 118

REST Call to Create an Application Network Profile, 118

Service Graph Example, 259

Set (Python), 99

String (Python), 99

Tenant Configuration—Complete, 120

Tenant Configuration in Cisco ACI Formatted in JSON, 94

Tenant Configuration in Cisco ACI Formatted in XML, 94

Tenant Creation, 218

Tenant mgmt Configuration for In-band Management, 222

Uplink Port Profile in Nexus 1000V, 156

Virtual Environment Creation, 102

XML Format, 111

YAML Format for Configuration Files, 95

YAML Libraries, 100

exception packets, 288

external networks, connecting to, 188, 240-241

F

fabric design (ACI)

benefits, 193

configuring, 219-235

- interface policy groups, 229*
- network management, 221-223*
- policy-based configuration of access ports, 223-228*
- PortChannels, 229-231*
- requirements, 219-220*
- virtual PortChannels (vPCs), 231-232*
- VMM domains, 233-235*
- zero-touch provisioning, 220-221*

forwarding, 194-202

- inside versus outside networks, 199-200*
- overlay frame format, 196*
- packet walk, 201-202*
- pervasive gateway, 198-199*
- prescriptive topology, 194-195*
- VXLAN forwarding, 197-198*

hardware and software requirements, 207-208

management model, 204-207

multitenancy, 218-219

- physical topology, 208-218
 - APIC design considerations*, 210-211
 - leaf design considerations*, 212-218
 - spine design considerations*, 211-212
- segmentation with endpoint groups, 202-204
- virtual topology configuration, 235-241
 - bridge domain*, 237-238
 - endpoint connectivity*, 238-240
 - external connectivity*, 240-241
- faults**, 274-279
- feed replication**, 19
- filters (APIC model)**, 67, 75
- FlexPod model**
 - design, 28
 - Vblock model versus, 27
- flooding mode**, 238
- flowlet switching**, 322-323
- 40-Gigabit Ethernet cabling**, 208
- forwarding extension**, 139
- forwarding in ACI fabric**, 194-202
 - inside versus outside networks, 199-200
 - overlay frame format, 196
 - packet walk, 201-202
 - pervasive gateway, 198-199
 - prescriptive topology, 194-195
 - unknown unicast and broadcast traffic, 213-214
 - VXLAN forwarding, 197-198
- function definition, Python**, 97-98

- functions**
 - abstract node functions, defining, 255-257
 - in service graph, 249-250
- fvCtx (private networks)**, 238

G

- Git/GitHub**, 92, 103-106
 - additional resources, 124
 - centralized versus distributed repositories, 104
 - commands in, 105-106
 - installing, 105
 - operations in, 104-105
 - version control terminology, 103-104
- glean packets**, 289
- goals of ACI (Application Centric Infrastructure)**, 248-249
- goodput**, 311
- graphical user interface (GUI) for APIC policy model**, 87
- Group Based Policy**, 190-191
- GUI (graphical user interface) for APIC policy model**, 87

H

- Hadoop**, 8
 - availability and resiliency, 12
 - burst handling and queue depth, 12-13
 - cluster design, 10
- hardware proxy**, 213, 237-238
- hardware requirements**
 - in ACI fabric, 207-208
 - service insertion, 247-248

head-of-line blocking (HOLB), 304
 health monitoring, 272-281

- events and logs, 279-280
- faults, 274-279
- health score, 280-281
- statistics, 273-274

 health score, 280-281
 health score logs, 280
 Heat, 174
 HFT (high frequency trading)

- topologies, 20

 HOLB (head-of-line blocking), 304
 Homebrew, 101
 Horizon, 174
 horizontal partitioning, sharding

- versus, 84

 host-port connectivity (APIC driver), 188
 hot migration, 134
 HPC (high-performance compute data centers), 14-15

- design requirements, 14-15
- design topologies, 15
- network requirements, 14
- QoS (quality of service), 312
- storage requirements, 14

 hybrid clouds, 38
 Hyper-V, 137-141, 162-163
 Hyper-V Switch, 138
 hypervisors, 128

- benefits, 179
- Cisco Nexus 1000V, 155-158
- Linux KVM, 141-149
- Microsoft Hyper-V, 137-141
- port extension with VN-TAG, 158-160
- VMware ESX/ESXi, 149-154

IaaS (Infrastructure as a Service), 38

- orchestrators, 47-56
 - Cisco Intelligent Automation for Cloud (CIAC), 52-54*
 - comparison of models, 55-56*
 - OpenStack, 48-51*. See also *OpenStack*
 - UCS Director, 51-52*
 - vCloud Director, 47*

 Icehouse, installing, 183-185
 imperative control models, 59
 importing modules, Python, 98
 infrastructure statistics, 273
 ingress leaf, policy enforcement, 67
 input queuing in crossbar fabrics, 303-304
 inside networks, outside networks

- versus, 199-200

 installers, OpenStack, 49-50
 installing

- Git/GitHub, 105
- Icehouse, 183-185
- Python packages, 101
- Python SDK for ACI, 122-123

 instances (AWS), 41
 interface policies, configuring, 228
 interface policy groups, configuring, 229
 interface profiles, creating, 226
 investment protection,
 Ironic, 174-175

J

JSON, 94-95, 100

K

KVM (Kernel-based Virtual Machine), 141-149

ACI integration, 163-164

bridging, 142-143

OVS (Open vSwitch), 143-149

server virtualization components,
128

L

labels (APIC model), 75, 78-79

latency

big data data centers, 13

metrics, 271-272

spine-leaf model, 35

ultra-low latency data centers (ULL),
16-20

Layer 3 networks, 61-62

Layer 4 through Layer 7 services.
See service insertion

leaf switches, 207

Cisco Nexus switches,

configuring switch profiles, 228

design considerations, 212-218

libraries (Python), 102

libvirt, 141-142

lifecycle of fault monitoring, 277

Linux KVM (Kernel-based Virtual Machine), 141-149

ACI integration, 163-164

bridging, 142-143

OVS (Open vSwitch), 143-149

server virtualization components,
128

listings. *See* examples

lists (Python), 98

**LND (Logical Network Definition),
140**

logical device selectors, 251,
264-265

logical devices, 250-251, 260-266

logical interfaces, naming, 250-251

**Logical Network Definition (LND),
140**

logical networks, 140

logical switches, 140

logs, health monitoring, 279-280

M

main() function, Python, 97

management model

in ACI fabric, 204-207

for service insertion, 245

management planes, 286-288

marking boundaries, 313-314

massively scalable data centers
(MSDC). *See* MSDC (massively
scalable data centers)

memory, centralized shared memory
(SoC), 306-309

metadevices, 254-255

Microsoft

Hyper-V, 137-141, 162-163

server virtualization components,
128

middle of row (MoR) model, 30

- ML2 (Modular Layer 2) plug-in, 180-181
- ml2_cisco_conf.ini file, 186-187
- ml2_conf.ini file, 186
- modeling tenants in XML, 119-120
- Modular Layer 2 (ML2) plug-in, 180-181
- modules, importing, Python, 98
- MoR (middle of row) model, 30
- MQC (Modular QoS CLI) model, 324-326
- MSDC (massively scalable data centers), 21-25
 - design requirements, 24
 - design topologies, 25
 - network requirements, 23-24
 - QoS (quality of service), 312
 - storage requirements, 24
 - system characteristics, 22
- multicast, VXLANs without, 137
- multicast policy enforcement, 69-70
- multicast switching over crossbar fabrics, 298
- multi-hypervisor support (OpenStack), 49
- multistage crossbar fabrics, 305-306
- multistage SoC (centralized shared memory), 306-309
- multitenancy in ACI fabric, 218-219

N

- namespaces, 215-216
- naming conventions
 - ACI, 113
 - logical interfaces, 250-251
 - service insertion configuration, 265-266

- NETCONF (Network Configuration Protocol), REST and SNMP versus, 92
- network interface cards (NICs), virtual network adapters, 132
- network latency, big data data centers, 13
- network management options, 92-93, 221-223
- network requirements
 - big data data centers, 9
 - cloud computing, 39-41
 - high-performance compute data centers (HPC), 14
 - massively scalable data centers (MSDC), 23-24
 - ultra-low latency data centers (ULL), 17-18
 - virtualized data centers, 6
- network sites, 140
- network virtualization, 5
- Neutron, 169-172
- neutron.conf file, 186
- NIC teaming, 150
- NICs (network interface cards), virtual network adapters, 132
- NoSQL, 8-9
- Nova, 168-169

O

- object store (ACI), 114
- object tree (ACI), 108-114
 - classes and relationships, 109-113
 - naming conventions, 113
 - object store, 114
- Observer, 82
- Open vSwitch Database (OVSDB), 149

Open vSwitch (OVS), 143-149

- architecture, 143-145
- example topology, 145-146
- OpenFlow, 147-149
- OpenStack, 146
- OpenStack driver, 180
- OVSDB (Open vSwitch Database), 149

OpenFlow, 147-149**OpenStack, 48-51**

- ACI (Application Centric Infrastructure) and
 - benefits, 177-180*
 - features and capabilities, 191*
 - integration, 180-181*
- ACI integration, 163-164
- architecture models, 50-51
- components, 167-168
 - Cinder, 173*
 - Heat, 174*
 - Horizon, 174*
 - Ironic, 174-175*
 - Neutron, 169-172*
 - Nova, 168-169*
 - Swift, 173*
- deployment, 176-177
 - configuring APIC driver, 185-188*
 - example, 182-189*
 - installing Icehouse, 183-185*
 - troubleshooting, 188-189*
- Group Based Policy, 190-191
- installers, 49-50
- multi-hypervisor support, 49
- networking considerations, 51
- OVS (Open vSwitch), 146
- projects and releases, 48-49

operating systems, ACI Fabric OS, 79-80

operational model (ACI), 91

- additional resources, 124
- APIC interfaces, 106-108
- Git/GitHub, 103-106
 - centralized versus distributed repositories, 104*
 - commands in, 105-106*
 - installing, 105*
 - operations in, 104-105*
 - version control terminology, 103-104*

JSON, 94-95

network management options, 92-93

object tree, 108-114

classes and relationships, 109-113

naming conventions, 113

object store, 114

Python, 96-103

characteristics of, 96-97

data structures, 98-99

function definition, 97-98

installing packages, 101

main() function, 97

online tutorial, 96

package requirements, 101-102

parsing files, 99-100

running, 101

verifying scripts, 101

virtual environments, 102-103

Python SDK for ACI, 122-124

developing Python scripts, 123-124

finding Python scripts, 124

installing, 122-123

- REST calls, 114-122
 - any-to-any policy*, 121-122
 - contracts*, 120-121
 - modeling tenants in XML*, 119-120
 - sending*, 115-117
 - syntax*, 117-119
- REST protocol, 93-94
- XML, 94
- YAML, 95-96
- orchestration**, 5
- orchestrators (IaaS)**, 47-56
 - Cisco Intelligent Automation for Cloud (CIAC), 52-54
 - comparison of models, 55-56
 - OpenStack, 48-51
 - UCS Director, 51-52
 - vCloud Director, 47
- OS deployment**, 44-47
- output queuing in crossbar fabrics**, 302-303
- outside networks, inside networks versus**, 199-200
- overlay frame format in ACI fabric**, 196
- overlay normalization**, 160-161
- overspeed in crossbar fabrics**, 298
- oversubscription ratio, big data data centers**, 13
- OVS (Open vSwitch)**, 143-149
 - architecture, 143-145
 - example topology, 145-146
 - OpenFlow, 147-149
 - OpenStack, 146
 - OpenStack driver, 180
 - OVSDB (Open vSwitch Database), 149
- OVSDB (Open vSwitch Database)**, 149

P

- PaaS (Platform as a Service)**, 38
- packages (Python)**
 - for ACI SDK, installing, 122-123
 - installing, 101
 - requirements, 101-102
- packet forwarding, VXLANs**, 136-137
- packet traffic**, 155, 201-202
- packets**
 - classification, 290
 - rate-controlling mechanisms, 290-291
 - types of, 288-290
- paravirtualization**, 138
- parsing files, Python**, 99-100
- partitioning, sharding versus**, 84
- path statistics**, 273
- performance, spine-leaf model**, 35
- pervasive gateway in ACI fabric**, 198-199
- PFC (priority flow control)**, 318-319
- physical domains**, 216
- physical servers, connecting to**, 239
- physical topology**
 - ACI as,
 - in ACI fabric, 208-218
 - APIC design considerations*, 210-211
 - leaf design considerations*, 212-218
 - spine design considerations*, 211-212
- pip**, 101
- Platform as a Service (PaaS)**, 38
- POD model**, 26-28

Policy Driven Data Center
 need for, 57-59
 promise theory, 59-61
policy groups, configuring, 229
Policy Manager, 81
policy models. *See* APIC (Application Policy Infrastructure Controller) model
policy resolution immediacy, 162
port extension with VN-TAG, 158-160
port profiles, 156-158
PortChannel, configuring, 188, 229-231
ports, configuring, 223-228
Postman, 94, 115
Precision Time Protocol (PTP), 271-272
prescriptive topology in ACI fabric, 194-195
preventing attacks, CoPP (Control Plane Policing), 288-291
priority flow control (PFC), 318-319
priority queues, 321-322
private clouds, 38
private Layer 3 networks, 61-62
project requirements, data center architecture designs, 29
projects (OpenStack), 48-49
promise theory, 59-61
PTP (Precision Time Protocol), 271-272
public clouds, 38
Puppet, 46-47
PXE booting, 43-44
Python, 92, 96-103
 additional resources, 124
 characteristics of, 96-97

data structures, 98-99
 function definition, 97-98
 installing packages, 101
 main() function, 97
 online tutorial, 96
 package requirements, 101-102
 parsing files, 99-100
 running, 101
 sending REST calls, 115-116
 verifying scripts, 101
 virtual environments, 102-103
Python SDK for ACI, 122-124
 developing Python scripts, 123-124
 finding Python scripts, 124
 installing, 122-123

Q

qemu, 141
QoS (quality of service), 309
 buffer bloat, 317-318
 buffering, explained, 315-317
 Data Center Bridging Exchange (DCBX), 320
 Data Center TCP (DCTCP), 320-321
 Early Congestion Notification (ECN), 320-321
 Enhanced Transmission Selection (ETS), 319
 flowlet switching, 322-323
 implementation in Cisco Nexus switches, 324-326
 priority flow control (PFC), 318-319
 priority queues, 321-322
 requirements, 309-314
Quantum. *See* Neutron
queue depth, big data data centers, 12-13

R

- raised (fault monitoring lifecycle), 278
- raised-clearing (fault monitoring lifecycle), 279
- rate-controlling mechanisms for packets, 290-291
- RBAC (role-based administrative control) for APIC policy model, 88-89
- receive packets, 288
- redirected packets, 289
- redundancy, spine-leaf model, 34-35
- regions (AWS), 41
- relationships (ACI), 109-113
- releases (OpenStack), 48-49
- reliability
 - replication and, 84
 - sharding and, 85
- rendering service graph, 246-249
- replication, reliability and, 84
- requirements
 - configuring Cisco ACI fabric, 219-220
 - design requirements
 - big data data centers*, 11-13
 - high-performance compute data centers (HPC)*, 14-15
 - massively scalable data centers (MSDC)*, 24
 - ultra-low latency data centers (ULL)*, 18-19
 - virtualized data centers*, 6
 - hardware and software requirements
 - in ACI fabric*, 207-208
 - service insertion*, 247-248
 - network requirements
 - big data data centers*, 9
 - cloud computing*, 39-41
 - high-performance compute data centers (HPC)*, 14
 - massively scalable data centers (MSDC)*, 23-24
 - ultra-low latency data centers (ULL)*, 17-18
 - virtualized data centers*, 6
 - packages (Python), 101-102
 - project requirements, data center architecture designs, 29
 - QoS (quality of service), 309-314
 - storage requirements
 - big data data centers*, 11
 - high-performance compute data centers (HPC)*, 14
 - massively scalable data centers (MSDC)*, 24
 - ultra-low latency data centers (ULL)*, 18
 - virtualized data centers*, 7
 - switch architecture, 291
 - traffic requirements, VMware ESX/ESXi, 151-152
- resiliency, **big data data centers**, 12
- REST calls, 114-122
 - any-to-any policy, 121-122
 - contracts, 120-121
 - modeling tenants in XML, 119-120
 - sending, 115-117
 - syntax, 117-119
- RESTful API
 - for APIC policy model, 88
 - defined, 92
 - implementation, 93-94
 - NETCONF and SNMP versus, 92

retaining (fault monitoring lifecycle),
279

RFC 3535, 91, 124

RFC 4594, 310

role-based administrative control
(RBAC) for APIC policy model,
88-89

running Python, 101

S

S3 (Amazon Simple Storage Service),
42

SaaS (Software as a Service), 38

scalability, spine-leaf model, 35

scheduler in crossbar fabrics, 301

SCVMM (System Center Virtual
Machine Manager), 138

SCVMM Server Console, 138

segmentation

with endpoint groups, 202-204

in virtualized data centers

VLANs, 134, 214-215

VXLANs, 134-137

sending REST calls, 115-117

sEPG (source EPGs), 66

servers

physical servers, connecting to, 239

provisioning, automating, 43

OS deployment, 44-47

PXE booting, 43-44

virtual servers, connecting to,
239-240

virtualization, 3

*ACI (Application Centric
Infrastructure) modeling of*,
160-165

components, 128

distributed switching, 133

EPGs (endpoint groups), 133

hot migration, 134

service insertion, 245

virtual network adapters, 132

virtual software switches,
129-133

service chaining, 179

service delivery model, 37

service graph, 243

configuring, 252-266

abstract node connectors,
257-258

abstract node elements, 258

abstract node functions,
255-257

*concrete and logical device
definition*, 260-266

connecting abstract nodes,
258-260

metadevices, 254-255

naming conventions, 265-266

service graph boundaries, 253

connecting EPGs with, 244-245

defining, 249-250

defining configuration, 245

rendering, 246-249

service insertion, 243

benefits, 244

concrete and logical devices,
250-251

configuring, 252-266

abstract node connectors,
257-258

abstract node elements, 258

abstract node functions,
255-257

- concrete and logical device definition*, 260-266
- connecting abstract nodes*, 258-260
- metadevices*, 254-255
- naming conventions*, 265-266
- service graph boundaries*, 253
- connecting EPGs with service graph, 244-245
- defining service graph, 249-250
- hardware and software requirements, 247-248
- logical device selectors, 251
- management model, 245
- rendering service graph, 246-249
- splitting bridge domains, 251
 - for virtualized servers, 245
- services virtualization, 5
- sets (Python), 98-99
- setup tools (Python), 101
- sharding, 84-87
 - horizontal partitioning versus, 84
 - reliability and, 85
 - technology, 86-87
- shared infrastructure, POD model, 26-28
- show-tech output, 281
- single-homed servers design, 32
- SNMP (Simple Network Management Protocol), REST and NETCONF versus, 92
- soaking (fault monitoring lifecycle), 278
- soaking-clearing (fault monitoring lifecycle), 278
- SoC (centralized shared memory), 306-309
- Software as a Service (SaaS), 38
- software requirements
 - in ACI fabric, 207-208
 - service insertion, 247-248
- source EPGs (sEPG), 66
- spine switches, 207
 - design considerations, 211-212
- spine-leaf model, 33-35
 - in ACI fabric, 208-218
 - APIC design considerations*, 210-211
 - leaf design considerations*, 212-218
 - spine design considerations*, 211-212
 - redundancy, 34-35
 - scalability and performance, 35
- splitting bridge domains, 251
- SQL database farm example (APIC model), 76-79
- statistics, health monitoring, 273-274
- storage requirements
 - big data data centers, 11
 - high-performance compute data centers (HPC), 14
 - massively scalable data centers (MSDC), 24
 - ultra-low latency data centers (ULL), 18
 - virtualized data centers, 7
- storage virtualization, 4-5
- store-and-forward switching, cut-through switching versus, 292-295
- strings (Python), 98-99
- subjects (APIC model), 73-75
- superframing in crossbar fabrics, 299-301
- Swift, 173

switch architecture

- centralized shared memory (SoC), 306-309
- by Cisco Nexus model, 326
- CoPP (Control Plane Policing), 288-291
- crossbar switch fabric architecture, 295-306
 - benefits*, 297
 - cut-through switching*, 301-302
 - HOLB (head-of-line blocking)*, 304
 - input queuing*, 303-304
 - multicast switching*, 298
 - multistage crossbar fabrics*, 305-306
 - output queuing*, 302-303
 - overspeed*, 298
 - scheduler*, 301
 - superframing*, 299-301
 - unicast switching*, 297
 - VoQ (virtual output queuing)*, 304
- cut-through switching, 292-295
- data, control, management planes
 - interaction*, 287-288
 - separation*, 286
- requirements, 291
- summary of, 291

switch profiles, configuring, 228**System Center Virtual Machine Manager (SCVMM), 138****T****taboos (APIC model), 74-75****TCAM (ternary content-addressable memory), 65****telemetry, 179-180, 267**

- atomic counters, 267-270
- health monitoring, 272-281
 - events and logs*, 279-280
 - faults*, 274-279
 - health score*, 280-281
 - statistics*, 273-274
- latency metrics, 271-272
- show-tech output, 281

Tenant ID, 138**tenants (APIC model), 61-63**

- modeling in XML, 119-120
- multitenancy, 218-219
- statistics, 273

10-Gigabit Ethernet cabling, 208**ternary content-addressable memory (TCAM), 65****three-tier topology, 1-2****time to live (TTL) attacks, 289-290****top of rack (ToR) model, 30-32****topologies**

- high-performance compute data centers (HPC), 15
- massively scalable data centers (MSDC), 25
- physical topology in ACI fabric, 208-218
- prescriptive topology in ACI fabric, 194-195
- summary of, 25
- ultra-low latency data centers (ULL), 19-20
- virtual topology, configuring, 235-241

Topology Manager, 81-82**ToR (top of rack) model, 30-32****traffic requirements, VMware ESX/ESXi, 151-152**

troubleshooting. *See also* telemetry
 levels of, 274
 OpenStack deployment, 188-189
 trust boundaries, 313-314
 TTL (time to live) attacks, 289-290
 tunneling, 179
 tuples (Python), 98

U

UCS Director, 51-52
 ULL (ultra-low latency data centers),
 16-20
 design requirements, 18-19
 design topologies, 19-20
 network requirements, 17-18
 QoS (quality of service), 312
 storage requirements, 18
 unicast policy enforcement, 66-68
 unicast switching over crossbar
 fabrics, 297
 unknown unicast traffic, forwarding,
 213-214
 user interface for APIC policy model
 CLI (command-line interface), 87
 GUI (graphical user interface), 87
 RESTful API, 88
 strengths and weaknesses, 106-108

V

vApp, 150-154
 Vblock model, FlexPod model versus,
 27
 vCenter, 149
 vCloud Director, 47, 149, 152-154
 verifying Python scripts, 101

version control, Git/GitHub, 103-106
 centralized versus distributed
 repositories, 104
 commands in, 105-106
 installing, 105
 operations in, 104-105
 version control terminology,
 103-104
 video, QoS (quality of service),
 309-310
 virt-install, 142
 virt-manager, 142
 virtual domains, 216-217
 Virtual Env, 102-103
 virtual environments, 102-103, 123
 Virtual Machine Management Service
 (VMMS), 138
 Virtual Machine Manager (VMM)
 domains, 161-162, 233-235
 virtual machine managers, 128
 virtual network adapters, 132
 virtual output queuing (VoQ), 304
 virtual PortChannels (vPCs),
 configuring, 231-232
 virtual servers, connecting to,
 239-240
 virtual software switches, 128, 133
 Cisco Nexus 1000V, 155-158
 OVS (Open vSwitch), 143-149
 reasons for, 129-131
 vSwitch and distributed virtual
 switches, 150-151
 Virtual Subnet Identifier (VSID), 138
 virtual topology, configuring,
 235-241
 bridge domain, 237-238
 endpoint connectivity, 238-240
 external connectivity, 240-241

virtualized data centers, 2-7

challenges, 127

hypervisors

*benefits, 179**Cisco Nexus 1000V, 155-158**Linux KVM, 141-149**Microsoft Hyper-V, 137-141**port extension with VN-TAG,
158-160**VMware ESX/ESXi, 149-154*

integration approaches, 127

network and design requirements, 6

network virtualization, 5

orchestration, 5

QoS (quality of service), 312

segmentation

VLANs, 134

VXLANS, 134-137

server virtualization, 3

*ACI (Application Centric
Infrastructure) modeling of,
160-165**components, 128**distributed switching, 133**EPGs (endpoint groups), 133**hot migration, 134**service insertion, 245**virtual network adapters, 132**virtual software switches,
129-133*

services virtualization, 5

storage requirements, 7

storage virtualization, 4-5

virt-viewer, 142**Visore, 108****VLANs**

EPGs (endpoint groups) versus, 65

namespaces, 215-216

segmentation, 134, 214-215

VMM (Virtual Machine Manager)

domains, 161-162, 233-235

VMM Manager, 83**VMMS (Virtual Machine
Management Service), 138****VMware**

ESX/ESXi, 149-154

*ACI integration, 164-165**traffic requirements, 151-152**vCloud Director and vApp,
152-154**vSwitch and distributed virtual
switches, 150-151*server virtualization components,
128**VN networks, 140****vNetwork Distributed Switch, 149****VN-TAG, 158-160**voice, QoS (quality of service),
309-310**VoQ (virtual output queuing), 304****VPC (Amazon Virtual Private Cloud),
43**vPCs (virtual PortChannels),
configuring, 231-232**vShield Manager, 149-152****VSID (Virtual Subnet Identifier), 138****vSphere ESXi, 149****vSwitches, 150-151****VXLANS**

forwarding, 197-198

namespaces, 215-216

overlay frame format, 196

packet format, 135
packet forwarding, 136-137
segmentation, 134-137
vShield Manager, 151-152
without multicast, 137

W

Websockets, 101
whitelist model (taboos), 75
Windows Azure Pack, 162-163
WMI (Windows Management
Instrumentation), 138
WNV (Windows Network
Virtualization), 138

X

Xcode, 101
XEN, 128
XML, 94, 119-120

Y

YAML, 95-96, 99-100

Z

zero-touch provisioning, 220-221

This page intentionally left blank

PEARSON IT CERTIFICATION

Browse by Exams ▾

Browse by Technology ▾

Browse by Format

Explore ▾

I'm New Here - Help!

Store

Forums

Safari Books Online

Pearson IT Certification

THE LEADER IN IT CERTIFICATION LEARNING TOOLS

Visit pearsonITcertification.com today to find:

- IT CERTIFICATION EXAM information and guidance for



CompTIA

Microsoft

vmware

Pearson is the official publisher of Cisco Press, IBM Press, VMware Press and is a Platinum CompTIA Publishing Partner—CompTIA's highest partnership accreditation

- EXAM TIPS AND TRICKS from Pearson IT Certification's expert authors and industry experts, such as

- *Mark Edward Soper* – CompTIA
- *David Prowse* – CompTIA
- *Wendell Odom* – Cisco
- *Kevin Wallace* – Cisco and CompTIA
- *Shon Harris* – Security
- *Thomas Erl* – SOACP



- SPECIAL OFFERS – pearsonITcertification.com/promotions
- REGISTER your Pearson IT Certification products to access additional online material and receive a coupon to be used on your next purchase

Articles & Chapters



Blogs



Books



Cert Flash Cards Online



eBooks



Mobile Apps



Newsletters



Podcasts



Question of the Day



Rough Cuts



Short Cuts



Software Downloads



Videos



CONNECT WITH PEARSON IT CERTIFICATION

Be sure to create an account on pearsonITcertification.com and receive members-only offers and benefits

