



Virtual Routing in the Cloud

Exclusive Offer – 40% OFF

Cisco Press Video Training

livelessons®

ciscopress.com/video

Use coupon code CPVIDEO40 during checkout.



Video Instruction from Technology Experts



Advance Your Skills

Get started with fundamentals, become an expert, or get certified.



Train Anywhere

Train anywhere, at your own pace, on any device.



Learn

Learn from trusted author trainers published by Cisco Press.

Try Our Popular Video Training for FREE!

ciscopress.com/video

Explore hundreds of **FREE** video lessons from our growing library of Complete Video Courses, LiveLessons, networking talks, and workshops.

Cisco Press

ciscopress.com/video

Virtual Routing in the Cloud

Arvind Durai, CCIE No. 7016

Stephen Lynn, CCIE No. 5507 & CCDE No. 20130056

Amit Srivastava

Cisco Press

800 East 96th Street

Indianapolis, IN 46240 USA

Virtual Routing in the Cloud

Arvind Durai, CCIE No. 7016
Stephen Lynn, CCIE No. 5507 & CCDE No. 20130056
Amit Srivastava

Copyright© 2016 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

First Printing April 2016

Library of Congress Control Number: 2016934921

ISBN-13: 978-1-58714-494-3

ISBN-10: 1-58714-494-8

Warning and Disclaimer

This book is designed to provide information about CSR 1000V router and adoption of NFV technology in the cloud environment. 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.

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.

Special Sales

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

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

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

Feedback Information

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

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

We greatly appreciate your assistance.

| | |
|--|-------------------------------|
| Publisher | Paul Boger |
| Associate Publisher | Dave Dusthimer |
| Business Operation Manager, Cisco Press | Jan Cornelssen |
| Executive Editor | Brett Bartow |
| Managing Editor | Sandra Schroeder |
| Development Editor | Ellie Bru |
| Senior Project Editor | Tonya Simpson |
| Copy Editor | Kitty Wilson |
| Technical Editor(s) | Matt Bollick, Ray Wong |
| Editorial Assistant | Vanessa Evans |
| Cover Designer | Mark Shirar |
| Composition | Mary Sudul |
| Indexer | Brad Herriman |
| Proofreader | The Wordsmithery LLC |



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, CCRP, CCNA, CCNP, CCSP, CQVP, 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

Arvind Durai, CCIE No. 7016, is an advanced services principal architect for Cisco Systems. His primary responsibility in the past 17 years has been in supporting major Cisco customers in the enterprise sector, including financial, retail, manufacturing, e-commerce, state government, utility (smart grid networks), and health-care sectors. Some of his focuses have been on security, multicast, network virtualization, and data center, and he has authored several white papers and design guides on various technologies. He has also been involved in data center design for more than 10 years and has designed many enterprise private cloud data center environments.

Arvind maintains two CCIE certifications: Routing and Switching, and Security. He holds a Bachelor of Science degree in Electronics and Communication, a Master's degree in Electrical Engineering (MS), and a Master's degree in Business Administration (MBA). He is a coauthor of two Cisco press books, *Cisco Secure Firewall Services Module* and *TcL Scripting for Cisco IOS*.

He has coauthored IEEE WAN smart grid architecture and has been a panel member for IEEE publications. Arvind also has presented in many industry forums, such as IEEE and Cisco LIVE.

Stephen Lynn, CCIE No. 5507 (Routing & Switching/WAN/Security) and CCDE No. 20130056, is an architect at Cisco Systems in the U.S. federal area. He has been with Cisco for more than 16 years and is a subject matter expert on enterprise network architecture. His focus is on large-scale network designs, including campus, WAN, and data center. As a recognized expert within Cisco and in the industry, Stephen has been working on large-scale, complex wide-area network designs in an enterprise environment. Stephen's focus has been on architectural designs involving 1,000 nodes to more than 10,000 nodes, leveraging technologies such as DMVPN, GET VPN, and FlexVPN to provide transport encryption and network segmentation over IP transport such as MPLS/Ethernet. Other areas of focus include high availability and convergence, QoS, Performance Routing (PfR), and network virtualization.

Stephen is a well-known speaker who has presented at several conferences and seminars worldwide. He holds a Bachelor of Science in Electrical Engineering from the University of Virginia. Stephen is based out of the Cisco office in Washington, DC.

Amit Srivastava is a senior manager with Equinix, Inc. At Equinix his team is responsible for global network and product fulfillment for Equinix's Cloud Exchange platform. Amit formerly worked as a technical leader with Cisco Systems, Inc. He has developed, tested, and enhanced network software for nearly 14 years. Before joining Cisco, he held positions in software application development, management, and testing.

Amit was involved in developing embedded applications for mobile devices in his engagement with Hughes Networks prior to joining Cisco.

Amit has been involved in the development cycles of new operating systems such as IOS XR and IOS XE and delivering features such as MPLS-based Layer 2 and 3 VPNs and traffic engineering. With IOS XE, Amit has worked with platforms such as ASR 1000 and CSR 1000V right from their inception, delivering enterprise-level features like IPsec, NAT, firewalls, NetFlow, AVC, and QoS. Amit holds a Bachelor of Science degree in Electrical Engineering.

About the Technical Reviewers

Ray Wong is a technical marketing engineer (TME) for Cisco Systems. In his more than eight years with Cisco, he has worked in multiple roles, from system testing, to solution design and validation, to technical marketing. He was a major contributor in the Cisco Virtual Office (CVO) solution. Together with his TME role for Cisco Cloud Services Router (CSR 1000V), he is also a subject matter expert for IOS VPN, including DMVPN, GET VPN, and FlexVPN.

Ray holds a Bachelor of Science degree in Computer Science and Mathematics from the University of Wisconsin–Madison. He is also a frequent speaker at Cisco Live events.

Matt Bollick has worked in technical marketing at Cisco for the past 19 years, running an obstacle course of technologies, including SNA, ATM and Ethernet switching, service provider aggregation, metro Ethernet, network management, and enterprise branch architectures. He has also worked on a variety of products, including the Cisco 7500, 7200, LS1010, 8540, 7300, and Cisco 10K before finally settling down for the past several years as the platform architect for the ISR series of branch routers. In his spare time, Matt is an avid SCUBA diver in North Carolina.

Dedications

From Arvind:

I am thankful to God for everything. I would like to dedicate this book to my wife, Monica, and my son, Akhhill, who have been extremely patient and supportive during my long working hours. I am grateful to my parents for their blessings and for providing me with strong values.

I would also like to thank my parents-in-law, brother and family, and brother-in-law and family for all their support and wishes.

From Stephen:

I would like to dedicate this book to my wonderful and beautiful wife, Angela, and to my two incredible children, Christina and Ethan. Without your love, sacrifice, and support, this book would not have been possible. Thanks for putting up with the late nights and weekends I had to spend behind the computer and on conference calls instead of playing games, building Legos, and doing other fun family activities.

From Amit:

I would like to dedicate this book to my wife, Reshma, my daughter, Aarushi, and my parents. Without their love and support, I would never have been able to work on this. I would also like to thank my parents-in-law and my entire extended family. Their love and support have always been unconditional.

Acknowledgments

Arvind Durai:

Thanks to my wife, Monica, for encouraging me to write my third book. She inspired me and helped keep my spirits up all the time and provided her thoughts in multiple sections of this book. Thank you!!!

It was great working with Amit and Stephen. Their excellent technical knowledge and passion for writing made this writing experience a pleasure. I am looking forward to more years of working together as colleagues and friends.

Stephen Lynn:

A debt of gratitude goes to my coauthors, Arvind and Amit. Your knowledge and dedication to this project are appreciated more than you will ever know.

Acknowledgements for this book wouldn't be complete without mentioning my wife, Angela, who has endured and supported me through all my endeavors.

Amit Srivastava:

Special thanks to Arvind and Stephen, from whom I learned a lot while writing this book. I look forward to their continued support.

Our Acknowledgement

Many people within Cisco have provided feedback and suggestions to make this a great book. Thanks to all who have helped in the process, especially Ray Blair and Matt Falkner, for providing insightful input during the proposal process. A special thank you goes to our technical editors, Ray Wong and Matt Bollick, for your technical accuracy and insight into the technologies. Special thanks to Dimitris Vlassopoulos for providing his NSO lab setup and sharing his insights!

A big thank you goes out to the production team for this book, Brett Bartow, Ellie Bru, and Tonya Simpson, who have been incredibly professional and a pleasure to work with, and for making this book possible.

Contents at a Glance

| | | |
|------------|--|-----|
| | Introduction | xv |
| Chapter 1 | Introduction to Cloud | 1 |
| Chapter 2 | Software Evolution of the CSR 1000 | 37 |
| Chapter 3 | Hypervisor Considerations for the CSR | 59 |
| Chapter 4 | CSR 1000V Software Architecture | 95 |
| Chapter 5 | CSR 1000V Deployment Scenarios | 141 |
| Chapter 6 | CSR Cloud Deployment Scenarios | 185 |
| Chapter 7 | CSR in the SDN Framework | 223 |
| Chapter 8 | CSR 1000V Automation, Orchestration, and Troubleshooting | 247 |
| Appendix A | Sample Answer File for Packstack | 293 |
| | Index | 319 |

Reader Services

Register your copy at www.ciscopress.com/title/9781587144943 for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to www.ciscopress.com/register and log in or create an account*. Enter the product ISBN 9781587144943 and click Submit. Once the process is complete, you will find any available bonus content under Registered Products.

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

Contents

| | | |
|------------------|---|-----------|
| | Introduction | xv |
| Chapter 1 | Introduction to Cloud | 1 |
| | Evolution of the Data Center | 1 |
| | Data Center Architecture Building Blocks | 2 |
| | Introduction to Virtualization in the Data Center | 4 |
| | Evolution of Virtualization | 5 |
| | Conceptual Architecture of Virtualization | 5 |
| | Types of Virtualization Technologies | 6 |
| | <i>Server Virtualization</i> | 6 |
| | <i>Types of Server Virtualization</i> | 8 |
| | <i>Storage Virtualization</i> | 9 |
| | <i>Types of Storage Virtualization</i> | 11 |
| | <i>Network Virtualization</i> | 12 |
| | <i>Network Virtualization Evolution</i> | 13 |
| | <i>Types of Network Virtualization</i> | 14 |
| | <i>Service Virtualization</i> | 15 |
| | Introduction to the Multitenant Data Center | 16 |
| | Introduction to Cloud Services | 18 |
| | Infrastructure as a Service (IaaS) | 18 |
| | Platform as a Service (PaaS) | 19 |
| | Software as a Service (SaaS) | 20 |
| | Cloud Deployment Models | 20 |
| | Cloud Design Considerations | 21 |
| | <i>Domain 1: Infrastructure and Environmental</i> | <i>22</i> |
| | <i>Domain 2: Abstraction and Virtualization</i> | <i>23</i> |
| | <i>Domain 3: Automation and Orchestration</i> | <i>23</i> |
| | <i>Domain 4: Customer Interface</i> | <i>24</i> |
| | <i>Domains 5 and 6: Service Catalog and Financials</i> | <i>24</i> |
| | <i>Domains 7 and 8: Platform and Application</i> | <i>24</i> |
| | <i>Domain 9: Security and Compliance</i> | <i>24</i> |
| | <i>Domain 10: Organization, Governance, and Process</i> | <i>25</i> |
| | Enterprise Connectivity to the Cloud | 26 |
| | <i>Internet for Transport</i> | <i>26</i> |
| | <i>Direct Connectivity to a Cloud Provider</i> | <i>28</i> |
| | Enterprise Cloud Adoption Challenges | 29 |

- Software-Defined Networking 30
 - Open Networking Foundation 31
 - OpenDaylight Project 32
 - Network Function Virtualization 33
 - OpenStack 34
- Summary 35

Chapter 2 Software Evolution of the CSR 1000 37

- IOS Software Architecture 37
- IOS XE Architecture 39
 - The IOS XE Kernel 40
 - The IOS Daemon 40
 - The Forwarding Manager 41
 - The Interface Manager 41
 - The Platform Manager 41
- Cisco ASR 1000 System Architecture Overview 41
 - Route Processor 42
 - Embedded Service Processor 42
 - SPA Interface Processor 43
- Cloud Service Router 1000V Overview 44
- Deployment Requirements 45
- Elastic Performance and Scaling 47
- Rapid Deployment and Routing Flexibility in the Cloud 49
- CSR 1000V Deployment Examples 50
 - Secure Cloud VPN Gateway 50
 - Network Extension from Premises to Cloud 51
 - Segmentation Within a Cloud 52
- CSR 1000V Key Features 52
- Summary 57

Chapter 3 Hypervisor Considerations for the CSR 59

- Understanding Operating Systems 59
 - Operating System Design 60
 - Physical Resource Management* 60
 - Software Access to Physical Resources* 62
 - Kernels 63
 - Microkernels* 63

| | | |
|---------------------------|---|-----------|
| | <i>Hybrid Kernels</i> | 64 |
| | <i>The Cisco IOS Kernel</i> | 64 |
| | The Boot Process | 66 |
| | Linux Memory Management | 69 |
| | <i>Linux Swap Space and Memory Overcommit</i> | 69 |
| | <i>Linux Caching</i> | 71 |
| Understanding Hypervisors | | 71 |
| | How Does a Hypervisor Compare to an Operating System? | 72 |
| | Type 1 Hypervisor Design | 74 |
| | <i>Monolithic Architecture</i> | 74 |
| | <i>Microkernel Architecture</i> | 74 |
| | <i>Core Partitioning</i> | 75 |
| ESXi Hypervisor | | 75 |
| | Architectural Components of ESXi | 75 |
| | The VMkernel | 75 |
| | <i>Components of the VMkernel</i> | 76 |
| | Processes Running on the VMkernel | 77 |
| | Device Drivers | 78 |
| | File Systems | 79 |
| | Management | 80 |
| KVM | | 82 |
| | Architectural Components of KVM/QEMU | 84 |
| | Guest Emulator (QEMU) | 85 |
| | Management Daemon (Libvirt) | 88 |
| | <i>User Tools (virsh, virt-manager)</i> | 89 |
| Hyper-V | | 91 |
| Xen | | 92 |
| Summary | | 94 |
| Chapter 4 | CSR 1000V Software Architecture | 95 |
| | System Design | 95 |
| | Virtualizing the ASR 1001 into the CSR 1000V | 98 |
| | CSR 1000V Initialization Process | 99 |
| | CSR 1000V Data Plane Architecture | 100 |
| | CSR 1000V Software Crypto Engine | 103 |
| | Life of a Packet on a CSR 1000V: The Data Plane | 103 |
| | Netmap I/O | 104 |

- Packet Flow 106
- Device Initialization Flow* 106
- TX Flow* 107
- RX Flow* 108
- Unicast Traffic Packet Flow* 109

- Installing the CSR 1000V on a VMware Hypervisor 110
 - Bringing Up the VM with the CSR 1000V on ESXi 110
- Installing the CSR 1000V on a KVM Hypervisor 126
 - Bring Up the CSR 1000V as a Guest 126
- Performance Tuning of the CSR 1000V 137
- Summary 139

Chapter 5 CSR 1000V Deployment Scenarios 141

- VPN Services 141
 - Layer 2 VPNs 141
 - Layer 3 VPNs 142
 - Site-to-Site VPNs* 143
 - Remote Access VPNs* 147
- Use Cases for the CSR 1000V as a VPN Service Gateway 148
 - Enterprise Data Center Network Extension 148
 - The CSR 1000V as a VPN Gateway* 148
 - CSR for Secure Inter-Cloud Connectivity 152
 - Remote VPN Access into the Cloud 153
- BGP Route Reflector Use Case for the CSR 155
 - The CSR 1000V in a Hierarchical Route Reflector Use Case 157
- Planning for Future Branch Design with the CSR 1000V 162
 - Evolution of Branch Virtualization 164
- LISP and CSR 168
 - LISP Terminology 169
 - The LISP Data Plane 171
 - The LISP Control Plane 171
 - Typical LISP Use Cases 175
 - IP Mobility* 175
 - IPv6 Migration* 175
 - Network-to-Network Connectivity* 175
 - Network-to-Network Interconnection Topology and Configuration* 176
- Summary 183

| | | |
|------------------|---|------------|
| Chapter 6 | CSR Cloud Deployment Scenarios | 185 |
| | CSR in a Multitenant Data Center | 185 |
| | Cloudburst | 190 |
| | Direct Access Model | 191 |
| | Redirection Access Model | 192 |
| | The Cisco Inter-Cloud Fabric | 194 |
| | Private Cloud Deployment with CSR in OpenStack | 195 |
| | Introduction to OpenStack | 196 |
| | <i>Primary Use Case for OpenStack</i> | 196 |
| | <i>OpenStack Components</i> | 197 |
| | CSR Within OpenStack | 206 |
| | <i>CSR 1000V as a Neutron Router</i> | 206 |
| | <i>CSR 1000V as a Tenant Router</i> | 209 |
| | CSR 1000V in a Public Cloud | 211 |
| | Amazon Web Services Deployment for the CSR | 211 |
| | <i>Amazon Web Service Solutions</i> | 211 |
| | <i>Routing in AWS Clouds</i> | 212 |
| | CSR 1000V Deployment in AWS | 216 |
| | <i>Instantiate a CSR in AWS</i> | 217 |
| | Summary | 222 |
| Chapter 7 | CSR in the SDN Framework | 223 |
| | Deploying OpenStack | 225 |
| | CSR as an OpenStack Tenant Deployment | 235 |
| | Instantiate CSR Plugin to OpenStack | 242 |
| | Summary | 245 |
| Chapter 8 | CSR 1000V Automation, Orchestration, and Troubleshooting | 247 |
| | Automation | 248 |
| | BDEO | 248 |
| | NSO (Tail-f) | 249 |
| | <i>NSO Example for NFV Orchestration with OpenStack (Service Chain)</i> | 252 |
| | Orchestration | 267 |
| | Virtual Managed Services (VMS) | 267 |
| | Cisco Prime Network Services Controller (PNSC) | 269 |
| | CSR 1000V Troubleshooting | 271 |
| | Architecture Overview | 271 |

| | |
|--|-----|
| I/O Configuration | 272 |
| <i>vSwitch</i> | 272 |
| <i>PCI Passthrough</i> | 274 |
| <i>SR-IOV (Single Root I/O Virtualization)</i> | 274 |
| <i>Host Configurations</i> | 275 |
| Debugging Packet Loss | 276 |
| <i>High-Level Packet Flow</i> | 276 |
| <i>ESXi Packet Debugging</i> | 289 |
| Summary | 292 |

Appendix A Sample Answer File for Packstack 293

Index 319

Command Syntax Conventions

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

- **Boldface** indicates commands and keywords that are entered literally as shown. In actual configuration examples and output (not general command syntax), boldface indicates commands that are manually input by the user (such as a **show** command).
- *Italic* indicates arguments for which you supply actual values.
- Vertical bars (|) separate alternative, mutually exclusive elements.
- Square brackets ([]) indicate an optional element.
- Braces ({ }) indicate a required choice.
- Braces within brackets ({ [] }) indicate a required choice within an optional element.

Introduction

In today's business environment, enterprise customers are under more pressure than ever to innovate and adapt to new challenges and market conditions. Enterprises want to focus their investments on their core business while reducing IT spending.

The cloud offers enterprise customers many benefits, such as lower costs and flexibility. The cloud's elastic model enables a company to increase and decrease infrastructure capacity on demand. The usage-based model offered by the cloud helps governments and enterprises reduce costs while increasing business agility by moving applications to the cloud and consuming infrastructure resources from the cloud. This leads to enterprises looking at consuming network and IT services from the cloud rather than investing in in-house operations.

The enabling technology in unlocking the cloud is virtualization. Virtualization abstracts and isolates the computing hardware and underlying infrastructure into a logical resource pool, allowing key capabilities such as resource sharing, virtual machine (VM) isolation, and load balancing. These capabilities provide the fundamental building blocks for an agile and scalable cloud environment with rapid provisioning, workload sharing, and increased availability.

The surge in applications and IT service consumption moving to the cloud highlights the need for evolved technologies and network elements in the cloud that offer security and visibility to help businesses with performance and compliance verification. Evolved networks and network services enable the provider to offer cloud services with security, performance, and availability. The Cisco Cloud Services Router 1000V (CSR 1000V) is a fully virtualized software router that offers a platform for enterprises to extend the data center to the cloud and to enforce their policies in the cloud.

The Cisco CSR 1000V provides a transparent solution for extending IT services into provider-hosted clouds. The solution offers a rich set of features, including VPN, firewall, Network Address Translation (NAT), application visibility, and WAN optimization. These functions allow enterprise and cloud providers to build highly secure, scalable, and extensible cloud networks. In addition, the Cisco CSR 1000V supports a rich set of application programming interfaces (API), providing robust integration into software-defined networking (SDN) for automated provisioning of these networks and network services and allowing simplified management and orchestration, which help in driving down costs further.

Networks inherently carry vast amounts of information, including user locations, device capabilities, topologies, and end-to-end performance characteristics. When exposed appropriately through well-defined APIs, such information can be consumed by cloud applications to fine-tune and customize their efficient delivery. The future holds the promise of increasingly rich application–network interactions.

The primary objective of this book is to simplify design aspects and architectural details in a unified resource, augmenting Cisco's existing collection of installation and configuration guides for various cloud-related products and solutions. This book covers the key

virtualization technologies used in the cloud; it provides a concise, accessible presentation of cloud network services and the different types of operational environments in the cloud. Cloud networking service and delivery concepts are reinforced with illustrative examples; architecture of SDN orchestration and its connection to Cisco CSR 1000V network services are introduced and elaborated upon. In addition, the book reviews the building blocks of the CSR 1000V, covering its architecture and software design.

This book also explains network design and deployment scenarios for the Cisco CSR 1000V, which influence its pivotal role in the cloud environment. Furthermore, the book distills how intelligent networks help providers simplify cloud service management and reduce costs through efficient scaling and optimized capacity utilization. This book provides architectural knowledge that contextualizes the roles and capabilities of these advanced networks and network services, along with discussions of design factors essential for their insertion into cloud services:

- The book introduces the readers to the cloud and provides an overview to different types of cloud operational environments, including a prelude to the evolution of virtual routers.
- Virtualization is introduced as a pivotal technology in cloud adoption.
- The book covers the details of the operating systems and hypervisors on which virtual routers run. It provides details pertaining to the operational aspects of virtual routing.
- The reader is introduced to the architecture and software design of the Cisco CSR 1000V virtual router. The reader is subsequently introduced to a comprehensive set of APIs that can be leveraged by SDN.
- The book focuses on different designs and use cases and configuration examples for routing, secure extension of enterprises to the cloud, and VM mobility. It illustrates how the CSR 1000V addresses the challenges that an architect faces in migrating toward the cloud.
- This book covers the different management techniques available to simplify operational and monitoring aspects of cloud services.

Who Should Read This Book?

This book is targeted for a technical audience responsible for architecture, design, and deployment of data center and enterprise cloud services.

This book also caters to the next generation of cloud network operators to implement enterprise features in the cloud, leveraging the CSR 1000V.

After reading this book, you will have a better understanding of the following:

- Key virtualization concepts and cloud models
- CSR 1000V software architecture and design

- SDN and the CSR 1000V platform and API
- Simplification of data center multitenant design with the CSR 1000V
- Use cases for the CSR 1000V to simplify enterprise routing in the cloud
- Operational visibility, management, and control of an enterprise network in the cloud

How This Book Is Organized

This book is organized into the following chapters.

Chapter 1: Introduction to Cloud

This chapter introduces the concept of cloud computing. It describes the various cloud models available and how virtualization enables the present-day transition to the cloud. Multitenant data center designs are illustrated, and the concept of SDN is introduced here.

Chapter 2: Software Evolution of the CSR 1000

This chapter introduces the software evolution of the Cisco Cloud Services Router (CSR 1000V). It covers the infrastructure requirements and design considerations of a CSR 1000V, and it discusses the features that a CSR 1000V brings to the virtual routing realm.

Chapter 3: Hypervisor Considerations for the CSR

This chapter describes the different hypervisor technologies available on servers to manage the hardware resources for virtual machines. Hypervisor technology selection is an important consideration when deploying the CSR 1000V.

Chapter 4: CSR 1000V Software Architecture

This chapter describes the software design of the CSR 1000V. It details the control-plane and data-plane design of the CSR 1000V. It also describes licensing requirements, software implementation, and packet flow related to the CSR 1000V.

Chapter 5: CSR 1000V Deployment Scenarios

This chapter describes the common deployment scenarios for the CSR 1000V. It depicts these scenarios using configuration examples.

Chapter 6: CSR Cloud Deployment Scenarios

This chapter describes CSR 1000V deployments in the cloud and data center environments.

Chapter 7: CSR in the SDN Framework

This chapter describes SDN components. It also provides an overview of the CSR 1000V in the OpenStack framework. Case studies in this chapter aim to educate the reader on using the APIs for user-defined outcomes.

Chapter 8: CSR 1000V Automation, Orchestration, and Troubleshooting

This chapter provides an overview of CSR 1000V management tools for orchestration, monitoring, and troubleshooting. It also illustrates the operation workflow for deploying a CSR 1000V.

Introduction to Cloud

This chapter introduces the concept of cloud computing. It provides an overview of the evolution from the physical data center to the concept of virtualization within the data center. It describes the various cloud models available and how virtualization is enabling the present-day transition to the cloud. The chapter illustrates multitenant data center designs and introduces the concept of software-defined networking (SDN).

Evolution of the Data Center

A *data center* is a facility that houses computer systems and data storage. These systems are interconnected by high-speed network infrastructure to facilitate information dissemination and processing. The data center is in the center of the modern technology evolution and plays a key role in the paradigm shift in how the information and infrastructure system interact and integrate.

Traditionally, data centers have relied heavily on physical hardware. The physical size of a data center is defined by the infrastructure and the space in which the hardware is hosted. These characteristics in turn define the amount of data that can be stored and processed in that location.

In the early second half of the 20th century, the early predecessors of the data center were large mainframes that occupied entire rooms. The first mass-produced mainframe computer was delivered in the 1950s. This machine, the UNIVAC-I mainframe computer, was the size of a one-car garage 14 feet by 8 feet by 8.5 feet high, and it weighed 29,000 pounds. It could perform 1905 instructions per second. Despite its massive size, the UNIVAC was only a fraction as powerful as today's computers. The A9 processor in an iPhone 6s is more than 6 million (10^6) times more powerful than the UNIVAC. In the days when IT decisions revolved around the mainframe, everything from hardware, operating system, and applications that ran on top of it had to be made on an enterprise scale. However, all of these components ran within one device, offering limited scalability and flexibility.

During the 1980s, the computer industry experienced a transition that brought microcomputers and personal computers into wider use in business. This transition accelerated through the 1990s as microcomputers began filling the old mainframe computer rooms and functioning as servers. These rooms became known as data centers. Gone were the mainframes that filled entire rooms. In their place were rack-mountable servers.

In the early stage of data center evolution, enterprises built distributed data centers based on their business and IT requirements. As business opportunities continued to expand, enterprises required better data center services and holistic management strategy. The demands for higher scalability, increased uptime, and higher data bandwidth made it hard to satisfy business requirements with the distributed data center model.

These requirements drove enterprises to consolidate their data centers into centralized locations. They constructed dedicated facilities with dedicated cooling infrastructure, redundant power distribution equipment, and high-speed network fabric capable of supporting tens of thousands of servers to enable businesses to host a range of services. In the process of centralizing the data centers, distributed servers and storage devices were integrated, along with data and applications. The integration of the systems and data storage streamlined the infrastructure of the data center, allowing for greater sharing and utilization of system resources. The exercise also consolidated many arduous processes, such as system management, disaster recovery, and data backup, and contributed to enhanced IT services with unified management and increased business continuity.

The latter years of the 2000s saw a shift in the data center paradigm. Studies have shown that data center resources are underutilized; on average, these studies have found, data centers are only 20% utilized. Much of that underutilization is due to application silos with dedicated network, processing, and storage resources. Enterprises have been increasingly pressed to reduce IT spending while increasing business agility to launch new services. At the same time, users have increased expectations to be able to access information from anywhere, anytime, and using any device. These forces together have ushered in the era of cloud computing.

Today, data centers leverage virtualization technologies and cloud services to improve resource utilization and increase business agility and flexibility to quickly respond to rapidly changing business requirements. A variety of data center service models are available to enterprises. Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) are data center cloud offerings that offer different tiers of service-level considerations. After reading this chapter, you will have a better understanding of these concepts.

Data Center Architecture Building Blocks

Data center architecture should include a clear vision of the technology evolution in the future to avoid the entire facility being restricted by the least-scaling components in the building blocks. A typical data center consists of the following functional blocks:

- **Facility**—This block is the building that houses the data center. It provides robust and redundant power, cooling, and security for the data center. Power consumption and efficiency are top concerns facing data center managers because the power bill is generally the largest operating cost. It's also critical to ensure that a facility has sufficient power, cooling, and space for future growth. When data centers are built, they are designed for a specific power budget and cooling capacity. However, over time these needs tend to increase, and the data center runs out of capacity even though there is plenty of available space.
- **Network fabric**—The network fabric provides data transport and connectivity between entities within a data center. The fabric must have multiple-link redundancy and plenty of capacity for application traffic. It should offer simplified operation that has the complete view of the fabric. Going forward, services such as security and load balancing should be embedded into the fabric to increase the speed and simplify the management of these services.
- **Services**—This block is composed of network and security services, such as server load balancing, Secure Sockets Layer (SSL) or IPsec VPN offloading, application firewall, intrusion prevention, network analysis, and “packet capture” are examples of some of these services, although the list can change greatly depending on the particular requirements of the data center.
- **Compute**—This functional block includes servers and dedicated appliances, which are special types of single-purpose servers. This block typically includes rack-mounted or blade servers for efficient space, power, and cooling. These servers generally provide high performance, with plenty of CPU cores, and have as much memory as is practical to make the most efficient use of space.
- **Storage**—This functional block provides data storage services to the data center. The servers in the compute block either connect to a dedicated storage-area network (SAN) for block-based storage or use network-attached storage (NAS) systems for file-based storage.

Tip Raw volumes of storage are created in block-based storage, and each block can be controlled as an individual hard drive. Block-based storage can be used to store files, as well as special applications such as databases. This type of storage format is much more efficient and reliable than file-based storage.

File-based storage, the most commonly used type of storage, stores files and folders and provides the same visibility to all the users accessing the system. This type of storage is very inexpensive to maintain, and it's simple to integrate access control with a corporate directory. However, it is less efficient than block-based storage.

Traditional architectures for data centers leverage discrete tiers of servers for computing, where each tier uses dedicated servers to execute specific functions. The most well-known example is the three-tiered architecture, which consists of web, application, and database servers.

While the discrete multitiered architecture served as the de facto data center design in the early 2000s, rapid IT expansion and a desire to quickly roll out new applications highlight these shortfalls of the architecture:

- Lack of IT agility
- Scalability problems for virtualized workloads
- Inability to scale to new unstructured data environments

Tip *Unstructured data* refers to information that does not have a predefined data model. This type of data is often text centric but may include data such as dates, numbers, and facts. With a data structure that is not of a predefined format, it is hard for traditional programs to compare and extract the stored information. This unstructured data is very prevalent in the age of Facebook and other social media sites. Techniques such as data mining and text analytics are used to provide different methods for finding patterns in otherwise unstructured data sets.

Until very recently, storage, computing, and network resources were separate physical components that were managed separately. Even applications that interacted with these resources, such as system management and monitoring software, had different security and access policies.

Introduction to Virtualization in the Data Center

Virtualization is a word that has become synonymous with computers today. Given today's computing requirements, virtualization is an absolute necessity in the present-day work environment. Chances are that any person using a physical computer in today's world will be inconspicuously exposed to some form of virtualization.

As the word suggests, virtualization involves creating a virtual version of something that can be used without it being physically present where it is required. Take, for example, a server hosting a website. A small business owner, Bob, makes money by selling furniture online. He has a small server that can host his website and cater to a couple hundred hits—or shoppers accessing his website—per minute. With Thanksgiving coming, Bob expects his business to surge, along with the number of hits on his website. Today Bob has three options:

- He can buy a larger server for his website that, he is pretty sure, won't be required after the Thanksgiving rush.
- He can do nothing and allow his service to deteriorate during the peak season when his web users most need it.
- He can go online and rent a server for his computing needs for the duration he thinks it will be required. This server is actually a virtualized server sharing hardware

resources with virtual servers from other companies that are also trying to handle the Thanksgiving rush.

You don't have to be Sherlock to figure out the best option here. Virtualization is the best option for Bob. He can rent a server and share the compute resource for the duration of the surge. This way, he doesn't burn his dollars when his computing requirements decline, and he gives his users the service they need for the duration of the surge.

Evolution of Virtualization

Let's look at how our present-day virtualization technology evolved. During the 1950s and 1960s, the mainframe was the workhorse when it came to processing data or solving complex equations. Back in the day, the mainframe was one big computer that had all the answers. The world then moved on to a distributed architecture, with all computing requirements serviced by different components in the system. Computing, storage, and networking requirements were serviced by different entities. In some ways, virtualization technologies bring back many of the shared resource models from the mainframe days. Today your service provider virtualization cloud can cater to your computing, storage, and networking requirements. Years of evolution have brought us to a model that is actually similar to the mainframe architecture.

Here are a few benefits of virtualization:

- Optimal utilization of hardware resources so that there is no spare capacity sitting around unused
- Redundancy because virtual servers typically aren't tied to specific pieces of hardware that can fail
- Abstracted complexity so the virtual servers don't have to be virtualization aware
- Distributed storage, which offers high availability for data and faster failure recovery
- Cost-effectiveness because you pay for the capacity you need when you need it

Conceptual Architecture of Virtualization

As you look at the different virtualization technologies through the course of this chapter, you will find some uncanny resemblances between them. Figure 1-1 shows a conceptual overview of virtualization.

The lowest layer is the physical layer you wish to virtualize. It can be a server blade, a storage device, a network of storage devices, a network, or any physical entity you wish to virtualize. On top of that is a virtualization layer, which is a piece of software or hardware that controls access to the physical entities and presents these entities to the instances running on it to ensure efficient utilization of the hardware and fair distribution of resources. The virtual server, or virtual machine, instances run on top of this piece

of software or hardware, using the virtual resources presented to it by the virtualization layer. Some examples of virtual resources are network and storage virtual devices presented by the hypervisor to the virtual machine.

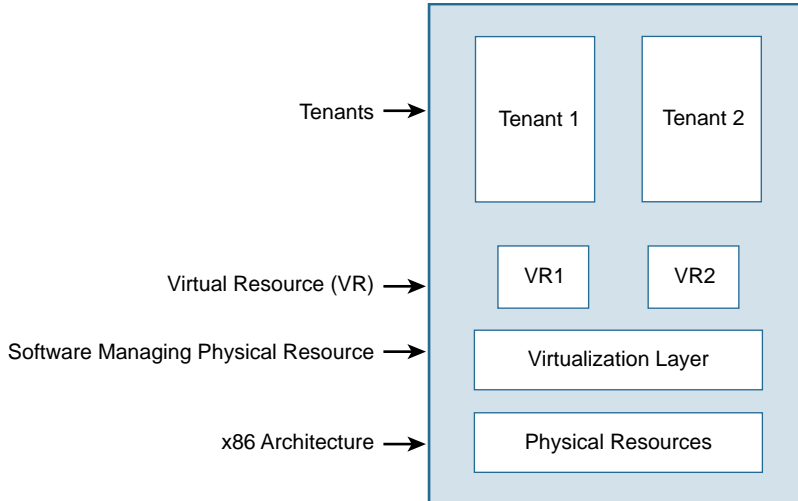


Figure 1-1 *Conceptual Overview of Virtualization*

Virtualization is a powerful concept. The lowest physical layer described might not be a single device. You can extrapolate the same concept to a group of devices that are networked together. You can also run virtualization with the virtualized instances (which is called *nested virtualization*). The core concepts, however, do not change.

Types of Virtualization Technologies

Virtualization is classified mainly based on what is virtualized or what it solves. There are three classification types:

- Server virtualization
- Storage virtualization
- Network virtualization

Server, storage, and network virtualization together constitute a data center service. Virtualization of these elements takes care of service virtualization.

Server Virtualization

The instruction sets for the x86 architecture were initially developed for embedded systems and small computers. However, this architecture grew in features and capability over the years. The original 32-bit architecture had a restriction of less than 4GB of memory per system. However, when AMD burst onto the scene with its Opteron 64-bit

version of the x86, it introduced essentially limitless RAM address space. 2^{64} bytes of RAM will give you memory on the order of 1 *billion* gigabytes. Throw in a 64-bit address bus, and you get much better I/O than with a 32-bit version.

Servers—or any hardware, for that matter—equipped with these processors are often underutilized if they run an application on a single operating system. Virtualization enables you to exploit the full capability of these modern-day masterpieces.

Without virtualization, an operating system manages the hardware and schedules it for applications. Software is tightly coupled to the hardware because all of the hardware resources are managed by a single OS. You run the applications that are supported by that particular OS only.

With virtualization, a hypervisor runs on the server and provides logical isolation between virtual instances sharing the same physical hardware. These virtual instances are logically isolated from one another and behave as if they are running on their own physical hardware. You run operating systems to manage an instance—that is, the view the hypervisor gives you of the hardware resource (memory, CPU, networks, and so on). Typically, *instance* is used as a synonym for a virtual machine.

With each virtual machine getting its share of the hypervisor-managed hardware resources, server virtualization ensures that you get the most out of your hardware.

The hypervisor virtualizes the x86 architecture. Memory, CPU, network cards, and other physical resources are presented to the instances running on the hypervisor. The instances get their share of the physical resource from the hypervisor (vMem, vCPU, vNIC, and so on). An instance runs an operating system (that is, a guest OS) using these virtual resources. Multiple such instances can run on a single hypervisor. Applications run on this guest OS as tenants, and the virtual resources given to this instance are scheduled by the guest OS. Figure 1-2 shows the components of server virtualization.

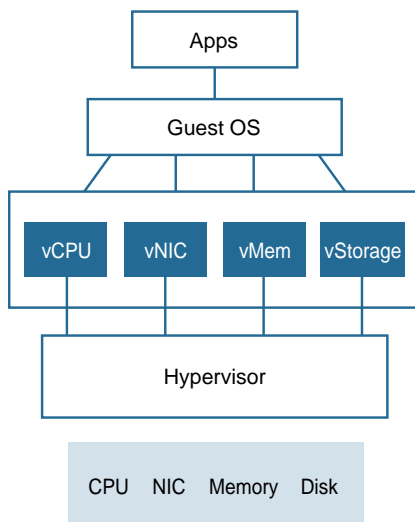


Figure 1-2 *Components of Server Virtualization*

Types of Server Virtualization

There are several different types of server virtualization:

- **Full virtualization**—With this server virtualization technology, the hypervisor does the heavy lifting. The hypervisor completely disassociates the guest OS from the physical machine, and the guest OS is not aware that it is running in a virtualized environment. The benefit of using this technology is that your guest OS doesn't need to be modified to run in such an environment.
- **Para-virtualization**—This server virtualization technology offers a set of software interfaces to the guest OS that are similar to the underlying physical server resource but not identical. The guest OS is aware of the fact that the hypervisor is presenting a virtual resource, and it is therefore referred to as an *enlightened guest*. A para-virtualized guest OS is a modified operating system that runs optimally on a hypervisor. However, this enlightenment means there is an inherent drawback to para-virtualized hypervisors: You must write drivers to enlighten the guest operating systems. However, the performance benefits of para-virtualization are immense. Many major operating system vendors have para-virtualization available today or planned for the near future to take advantage of the significant performance benefits it can provide. The Xen hypervisor is a prime example of para-virtualization. Figure 1-3 shows an enlightened guest where tenants are aware that they are running in a virtualized environment.

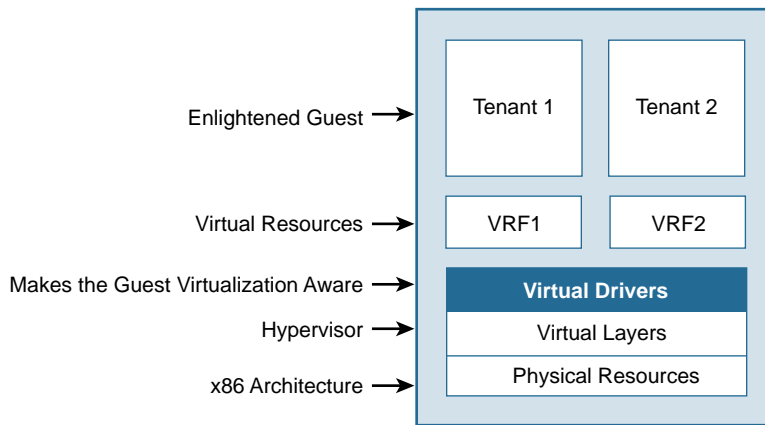


Figure 1-3 *Para-virtualization*

- **OS virtualization**—This server virtualization technology has no hypervisor at all. The host OS kernel allows multiple user spaces instead of one, thus providing isolation to each guest application. This concept is commonly referred to as containers. The upside of this approach is that the applications running within the containers use regular OS calls, with no application rewrite needed to work within such an environment. There does need to be some compatibility between the host and guest operating systems. Changes in either one can potentially break a working container. Figure 1-4 shows the elements of server virtualization.

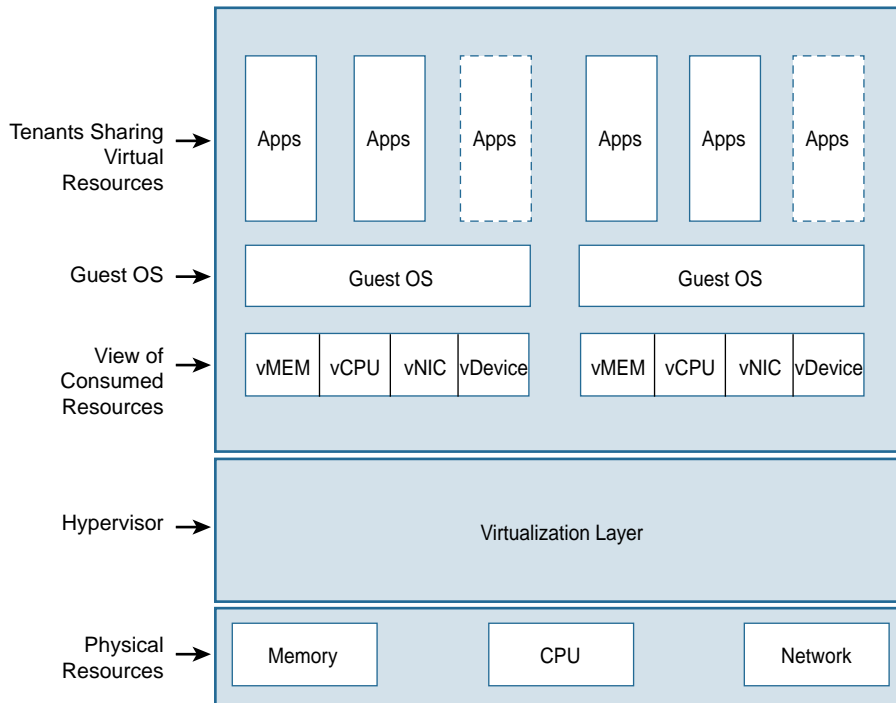


Figure 1-4 *Server Virtualization*

Chapter 3, “Hypervisor Considerations for the CSR,” covers in detail the different hypervisor technologies used for CSR in a virtualized environment.

Storage Virtualization

In its crudest form, storage involves a device connected to your computer storing all your data. The computer writes and retrieves data from this storage device. The amount of data that can be written onto the storage device is limited by the storage capacity of this device. Needless to say, your computer connected to this storage is dependent on it for all its storage requirements (assuming that the computer is not connected to any other storage network). It is evident that this basic form of storage has some shortcomings:

- Storage and scalability are limited.
- There is a single point of failure.
- The complexity of the storage media must be exposed to the applications/OS or the device controller.
- Only one computer can write data to or read data from a storage device.

Storage virtualization combines a bunch of networked storage devices into what appears to the users as a single storage entity. Specialized hardware or software manages the complexity of a storage network and gives each device wanting access to the storage a view of being directly connected to the storage media.

The Storage Networking Industry Association (SNIA) technical tutorial document defines storage virtualization as follows:

The act of hiding, abstracting or isolating the internal functions of a storage (sub) system or service from applications, host computers or general network resources for the purpose of enabling application and network-independent management of storage or data.

A storage system can be on a single disk or on an array of disks in any form, and specialized hardware or software manages the physical storage. For example, RAID (redundant array of inexpensive [or independent] disks) combines multiple physical disks into a single logical unit to be used by applications. This gives the user data redundancy because the data is stored on physically different devices. Two types of RAID are possible: hardware RAID and software RAID. With hardware RAID, you set up the multiple disks to function in a RAID configuration. The hardware RAID controller presents these multiple disks to the operating system as a single disk. The operating system is not aware of the multiple disks in RAID configuration. With software RAID, the OS does the heavy lifting. The operating system knows about the RAID storage and works accordingly.

A common storage virtualization mechanism is Network File System (NFS). NFS has been around since the mainframe era. The concept of NFS is simple: A storage repository on the network that can be accessed by computers on the network as if it were directly connected. NFS enables remote hosts to mount file systems over the network and interact with these file systems as though they were available locally. NFS is typically based on UDP (NFSv2 or older) or TCP (NFSv3). Using RPC (Remote Procedure Call), the NFS servers and clients communicate with each other, making the remote file system directly accessible to the local machine.

NFS and RAID are common examples of storage virtualization. Today, NFS and RAID are so popular that people use them without even being aware that they are using a form of storage virtualization. Figure 1-5 shows the components of storage virtualization.

Today, storage virtualization is commonly associated with a large SAN (storage area network) managed by a hardware or software virtualization layer that presents this vast storage as a single block to the servers requiring storage. The virtualization software and hardware sitting between the storage and the servers make the applications completely oblivious to where their data resides. This eases the administrative task because administrators can manage the storage as if it were an entity. This simplifies the operation and management of the storage system, offering scalability when there is a need for additional storage.

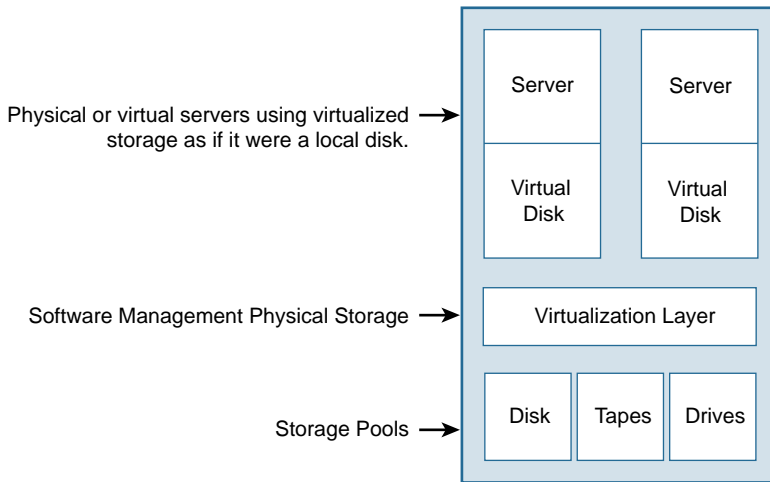


Figure 1-5 *Storage Virtualization*

Types of Storage Virtualization

Broadly speaking, we can categorize storage virtualization in three ways:

- **Host based**—With host-based virtualization, provisioning is done on the server to define logical volumes. Host-based virtualization presents the underlying storage to the operating system as a single disk. Regular device drivers manage the underlying physical storage, while the software layer above those drivers intercepts I/O calls and redirects them. Logical volume managers are examples of host-based virtualization.
- **Storage device based**—With this type, virtualization is brought to the storage hardware itself. One master device takes over all the I/O calls and redirects to other storage devices connected to it downstream. RAID is an example of storage device-based virtualization.
- **Network based**—With this type of virtualization, a network device sits between the host and the storage device that will virtualize and redirect I/O requests. The operating system is not aware of the virtualized storage resource it is using for storage. No changes need to be made in the operating system when working with network-based storage virtualization. The storage arrays and the network device that is virtualizing the I/O calls are able to communicate. Network-based virtualization combines the best features of host- and storage-based virtualization. The intelligence is centralized to a network device, such as a switch or a router, so the systems connected to the network can function independently and remain oblivious to the virtualization taking place. In addition, there is no need to tailor the OS/guest to make it work with network-virtualized storage.

As is the case with server virtualization, storage virtualization helps enterprises reap a number of benefits:

- **Hidden complexity**—With the virtualization layer abstracting the complexity, it becomes very easy for admins to provision new storage and/or migrate to new physical storage.
- **Thin provisioning**—Most storage virtualization solutions enable you to overcommit storage space with the help of thin provisioning. The concept enables operating systems to use only the physical storage they actually need for storing data rather than reserving all the potential storage they might need at any point in time. Thin provisioning is based on the presumption that not all applications will access the entire storage space allocated to them.
- **Performance and efficiency**—Virtualization enables you to write portions of your data across multiple storage devices simultaneously. This enhances the overall performance and efficiency of the system. Overall performance improves significantly, while reliability improves through the use of redundant physical devices.
- **Ease of management**—One management interface represents all storage devices; it can be very easy for network admins to provision, upgrade, and troubleshoot issues using this management interface.

Network Virtualization

Traditionally, networks have been meshes of routers and switches that forward data packets and offer services such as firewalls, access control, QoS, and security. Routers and switches are hardware devices that run specialized software. This combination of hardware and software provides network services and packet forwarding. But highly skilled engineers are needed to manage these resources and ensure that they run correctly.

Cisco and other network equipment manufacturers have long shipped routers and switches loaded with their proprietary software. The network engineering community felt the need to separate the control and data planes within this proprietary software architecture. This simply meant decoupling the intelligent (control plane) piece of software from the packet forwarding (data plane) logic. Because packet forwarding is a task that requires speed, it made sense to have it done in hardware, and changing this meant software could be separated within the router and switch operating systems. The control plane becomes a place where all the routing and switching decisions are made. Network devices have their data plane hardware programmed by the control plane to forward packets accordingly. This provided a good performance and manageability uplift to previous monolithic software architectures. Now a piece of software essentially programmed the hardware for network operations. This software to control hardware features was limited within a network device.

Well, why should the software to control the hardware features be limited to a single networking device? Why not do what the server and storage virtualization folks have

done and completely decouple the hardware from the service? This is exactly what network virtualization does: A piece of software manages the hardware and presents the applications' virtual resources and services derived from the hardware. Much as in server virtualization, the OS is presented with vMem, vCPU, and vNIC, and network virtualization creates virtual networks with vSwitches, vRouters, and virtual services like vFirewalls.

Server virtualization had a head start on network virtualization. It was conceived and implemented before network virtualization came in. With server virtualization, computing entered an era of seamless mobility, push-button recovery, snapshotting, and immensely utilized hardware and automation, to name a few goodies. While computing and storage gave their applications what they had always longed for, networks languished in the dark ages without virtualization.

Network virtualization is achieved by running a piece of software over the existing packet forwarding and software infrastructure and giving virtual network services to applications requiring them. One of the key factors that need to be considered while adopting the network virtualization solution is visibility of traffic flow via multiple systems and the capability to control a channel-based application's traffic flow. Development of these features was important for consumption of network virtualization.

Network Virtualization Evolution

It is important to understand that network virtualization did not just crop up recently. Server and storage virtualization have had a head start when it comes to virtualization at the macro level. Virtualizing the entire computing spectrum and entire storage areas have become very common. Although network virtualization has not reached the maturity of server/computing virtualization at a macro level, it is getting there. However, it is important to understand that virtualization is not new to the networking world. Features and functionalities of virtualization technologies have been around for a while. Network virtualization can be traced back to the early 1980s. There was need for scalable Ethernet networks to support rapidly expanding network growth, and there was a need to segment networks without having to use a router or a switch sitting at the edge of each segment. There was essentially a need for each department within a network to have its own broadcast domain. Having a router or a switch dedicated to each network segment was an expensive proposition, and the virtualized network segment was born. VLANs inserted a tag for each Ethernet packet, and switches had to handle packets of a particular tag and drop the rest. This meant that networks could be connected with as many spanning trees as there were tags available.

The VLAN was the first and is by far the most popular virtualization within a network. It was revolutionary in that it could create broadcast domains with a connected Ethernet circuit, and during the early 1990s it meant not buying expensive networking equipment.

In 1996, Ipsilon Networks introduced a flow management protocol. The Ipsilon product IP switch ATM 1600 used Asynchronous Transfer Mode (ATM) with IP routing. It faded away into oblivion. Cisco said "Why restrict this to ATM?" and went ahead with the proposal of tag switching. The Cisco proposal was originally called label switching,

and after the IETF standardized it, Multiprotocol Label Switching (MPLS) was born. MPLS was developed to give a switching flavor to routed packets and was envisioned to provide faster networks. However, it was empirically found not to have performance benefits. The value was (and still is) in the applications built around it, such as Layer 2 and 3 virtual private networks (VPN). MPLS provided a label-switched path within the IP network that helped build Layer 3 and 2 VPNs around it.

Within a Layer 3 router, there was a need for multiple routing tables. This not only provided additional security but also, to a certain extent, eliminated the need to have multiple routing devices for routing segmentation. Having multiple instances of a routing table within the same router is called *virtual routing and forwarding* (VRF). VRF with MPLS gives us our present-day Layer 2 and 3 VPNs.

With network virtualization, servers running virtual machines can now reap the benefits of virtualized networks, including automation, manageability, and decoupled software and hardware.

Types of Network Virtualization

We can classify network virtualization into two categories:

- **Protocol-based virtualization**—With this type of virtualization, a common shared network is segmented into multiple networks, using VRFs, VLANs, and VPN technologies such as MPLS L3VPNs and L2VPNs. Essentially, a private network is carved out of a shared network, using mechanisms that give the endpoints a private network over a shared network.

In general, if a shared medium is used as transport and networking protocol infrastructure to tunnel traffic across this shared medium, this would be called protocol-based network virtualization.

Protocol-based network virtualization involves the following design considerations:

- **Access control**—Access control involves giving the user access to a particular segment of the network based on credentials. Segmented areas of a network have different authentication mechanisms to allow different classes of user access.
- **Path isolation**—This is achieved by using VLANs, tunnels (GRE, IPsec, VPNs), VRF tables, and applications built around MPLS to isolate the paths within the shared network.
- **Services**—Network services (QoS, security, and so on) are provided based on who logged in.
- **Device-based virtualization**—This is a recent form of network virtualization and perhaps the more recognized one, too. Device-based virtualization involves running a hypervisor on a physical network (which is a mesh of routers, switches, and connection media—a traditional network) and virtualizing the entire network. This enables you to create network topologies in software, and the network hypervisor virtualizes the actual physical network and provides handles that can be used to

create network topologies. If the network device performs a specific network functionality (firewall, load balancer, VPN, and so on), this solution is referred to as network function virtualization (NFV).

Figure 1-6 shows the building blocks of network device virtualization.

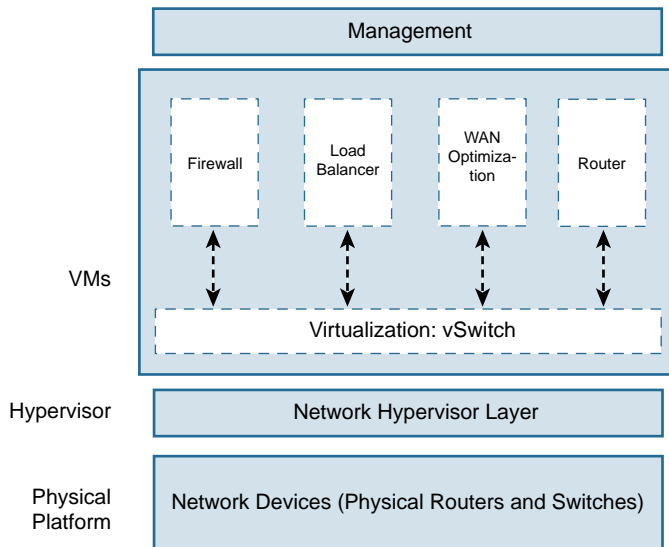


Figure 1-6 *Network Device Virtualization*

To achieve device-based virtualization, the design should consider the following:

- **Physical infrastructure**—The physical network here should be meshed. Routers and switches should have reachability to create this meshed network. The virtualization software uses a fully meshed infrastructure to deliver virtualized network functionality.
- **Network hypervisor**—The network-level hypervisor should abstract the physical infrastructure of network devices. It should be capable of providing virtual network devices to the software applications running on it.
- **Network applications**—Network applications utilize the virtualized network infrastructure provided to them by the network hypervisor. These provide the network admin with the necessary interface to connect the network elements together and build the topology.

Service Virtualization

Service virtualization is mainly used to virtualize an asset that is difficult to procure physically during the software development life cycle of a product. Development and testing teams can use this type of virtual asset instead of waiting for the actual physical resource to be available.

For example, consider a small software organization where the development and quality assurance teams are building an application that requires access to a mainframe in the back end. It is not cost-effective for the organization to get the engineers access to an actual mainframe. Even if the organization decides to procure a mainframe, engineering will have to wait until the resource is available and set up for them to use. The organization can instead use service virtualization to give the engineering team a virtual asset. This solution provides a number of benefits:

- It gives the organization a cost-effective way to procure the asset for its engineers.
- Engineering is more agile and does not have to wait for the asset to be made available.
- The service is available when needed.
- Downtime due to asset failure is minimized.
- The asset can more easily be centrally managed.

Keep in mind that service virtualization is not the same as network service virtualization. With service virtualization, the idea is to emulate a specific component within a modular application to provide software development and testing access to components that are required to test an application. Network service virtualization, on the other hand, involves offering a particular network service, such as a firewall, QoS, or DNS, as a service to a group of authenticated users.

Note A software router does not fit the device-based network virtualization category exactly. It is a network element but not a complete network. It actually better fits the server virtualization category because it involves virtualizing the server to provide virtual handles for memory, CPU, and I/O port, and running the router software over the virtualized resources.

As virtualization technologies mature and people get creative with deploying them, it will be increasingly difficult to fit technologies exactly into previously defined categories. The important thing here is to use the available virtualization tools to enhance a deployment and make it more cost-effective.

Introduction to the Multitenant Data Center

The term *multitenant data center* refers to a data center architecture that provides the capability to host multiple customers (tenants) and leverage common data center infrastructure and resources while providing logical and/or physical segmentation.

While multitenancy means that some infrastructure is shared, the degree of sharing and at what infrastructure layer depend on the type of service model. The highest degree of multitenancy sharing is SaaS. In this model, all customers are served from a single infrastructure, and every component is shared, all the way down to computing, storage,

application, and database. The degree of multitenancy in SaaS is different compared to PaaS and IaaS, where the sharing is at the infrastructure—that is, network, compute, and storage level—and not down to the application and database layer. You will learn about the details of SaaS, PaaS, and IaaS cloud infrastructures in the next section.

Multitenancy is based on virtualization technology. Virtualization technology enables multitenancy infrastructure and allows the creation of a virtual environment that provides logically separated systems on top of the common physical infrastructure.

The concept of multitenancy can be characterized at the different layers within data center functional blocks:

- Virtualization enables multiple virtual machines to operate and share resources on a physical machine.
- Virtualization provides logical abstraction to physical storage devices, allowing consistent presentation of storage resources.
- Network virtualization refers to the logical segmentation of network infrastructure on top of a common physical infrastructure. Each logical network is isolated from the others and provides privacy, security, and autonomy of forwarding policies.

The concept of virtualization is not new. In 1970, IBM introduced the mainframe virtualization system VM/370. It was one of the first examples of virtualization technology applied to a computing environment. The system provided multiuser access to seemingly separate and independent IBM VM/370 computing systems. Figure 1-7 shows IBM VM/370 virtualization. The control program (CP) together with the Conversational Monitor System (CMS) formed the virtual machine environment. CP was the resource manager of the system, and it was similar to the concept of today's hypervisor. The CP created virtual machines with guest operating systems.

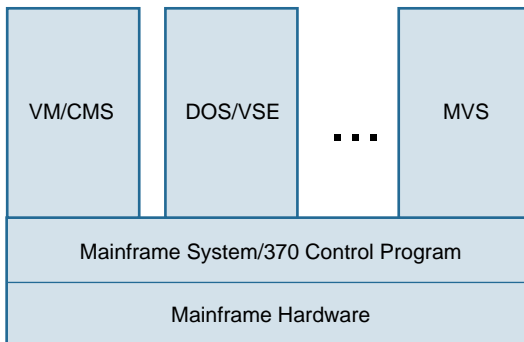


Figure 1-7 *Mainframe Virtualization*

Virtualization can extend beyond computing service and into the entire data center. Data center virtualization has enabled cost saving by reducing power consumption and cooling costs. At the same time, it has improved asset utilization by increasing efficiency

and availability to the resource workload. It reduces management touchpoints by taking advantage of a unified set of integrated management tools for physical, virtual, and cloud environments. Finally, it accelerates service delivery with IT agility and flexibility.

Introduction to Cloud Services

The following sections introduce the three offerings in the cloud: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Figure 1-8 provides a conceptual view of these services.

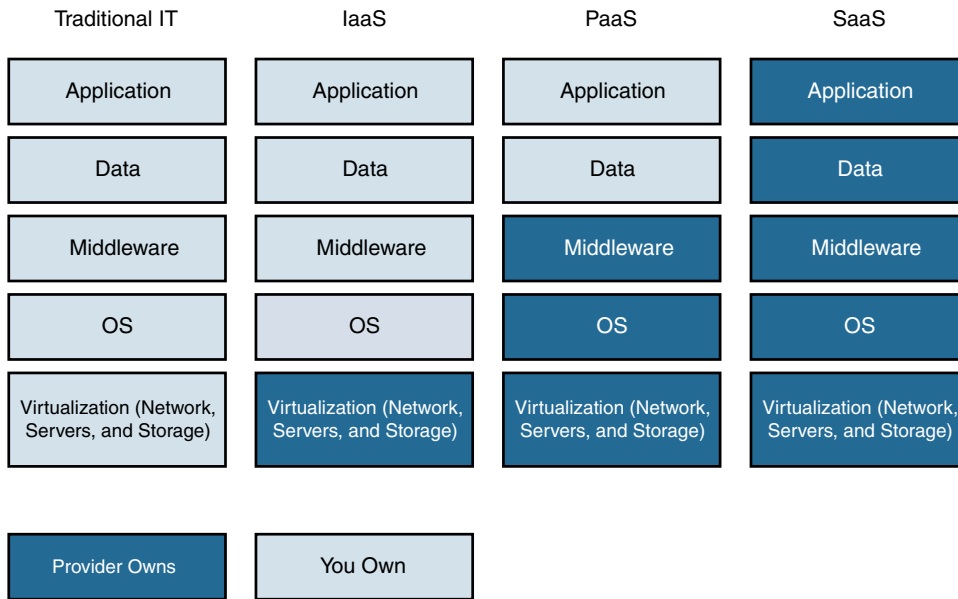


Figure 1-8 *Conceptual View of Cloud Services*

Infrastructure as a Service (IaaS)

IaaS enables a consumer to utilize storage, network, and computing resources. The consumer can manage these three components to deploy and run specific applications and software. In this way, the organization that consumes these resources from the cloud outsources the support operation for maintaining the IT infrastructure and the hardware cost and moves toward a “pay as you use” model.

An organization using IaaS still needs to have developers design applications and operations to manage operating systems in these hosted systems. The cost savings are huge because the organization doesn’t own the physical device. You can think of this as renting a home instead of buying it. The consumer’s rent is based on an on-demand model, and the service is provided based on a best-availability basis. With IaaS, it is possible to use a specific set of network, computing, and storage resources in the cloud

with a specific service level guarantee. These resources can be leveraged based on a contractual service level agreement (SLA) that is determined and negotiated between the consumer and provider. Companies tend to prefer this model because the end user experience can be managed within contractual boundaries. In this model, scaling can be provided within a limit for spurts in usage patterns.

The consumer using IaaS bears the licensing cost. The cost of using an IaaS infrastructure depends on the contractual usage of computing, memory, and storage. Another cost to be considered is data transfer to multiple instances for the same enterprise (IP address services, VPN services, security, monitoring, and so on).

Note The National Institute of Standards and Technology (NIST) defines IaaS as follows:

The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

Platform as a Service (PaaS)

PaaS abstracts many of the standard application stack-level functions and provides those functions as services. This enables programmers to leverage a tool set that is readily available in the cloud and lets them focus on the application logic rather than worry about the underlying infrastructure. PaaS provides programmers with libraries, services, and tools tied to the infrastructure of computing, network, and storage resources. Early players in the PaaS space were Google and Microsoft. Enterprises adopting PaaS should be careful of vendor lock-in. Back in the day, developers of Google apps had to code in Python, and Azure developers had to use .NET, which caused a vendor lock-in for the programmers.

Using PaaS, consumers can leverage middleware services without worrying about alignment with key hardware and software elements. A developer can get access to the complete stack of developer tools. PaaS in public cloud environments requires the clients to use the tool set offered by the provider for application development.

Note NIST defines PaaS as follows:

The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Software as a Service (SaaS)

SaaS enables consumers to utilize a hosted software licensing and delivery model. This model delivers an application as a service so that an organization can utilize its functions. SaaS is commonly used for non-core applications, such as WebEx and Salesforce. SaaS helps an organization offshore maintenance and reduces the initial capital expenditures on capability inception for non-core applications. Leveraging a provider model means an organization can use readily available applications without any investment in the initial infrastructure or licensing cost. This has helped IT departments provision collaboration and non-core application services to their users. Some of the key areas where this service is being utilized are customer relationship management (CRM) and corporate communication, such as WebEx and Cisco Hosted Collaboration Solution.

Note NIST defines SaaS as follows:

The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Cloud Deployment Models

Cloud in networking terms refers to storing and accessing data from an off-premise location. It provides agility for entities to scale and expand the IT infrastructure based on demand. This section talks more about characteristics of cloud deployment models.

Three cloud deployment models are common:

- **Public**—A public cloud is a multitenant environment that hosts multiple users who pay for the services they use. The users cannot see any of the other tenants utilizing the same environment.
- **Private**—The private cloud environment has automated provisioning of services for a single user in a hosted or on-premise location. The usage space is available to only a single organization. A private cloud reduces the regulatory risk around data security because it has a single-tenant deployment model.
- **Hybrid**—The hybrid cloud environment, as the name suggests, is a combination of private and public cloud environments. An organization that leverages two or more cloud infrastructures, regardless of whether they are private or public, needs to bind this infrastructure together with standard technology that enables application portability. By doing so, the organization gains elasticity of the public cloud infrastructure and can contain data ownership with security ingrained in the asset allocation model.

The three cloud models can be leveraged for the three service models described earlier as shown in Figure 1-9.

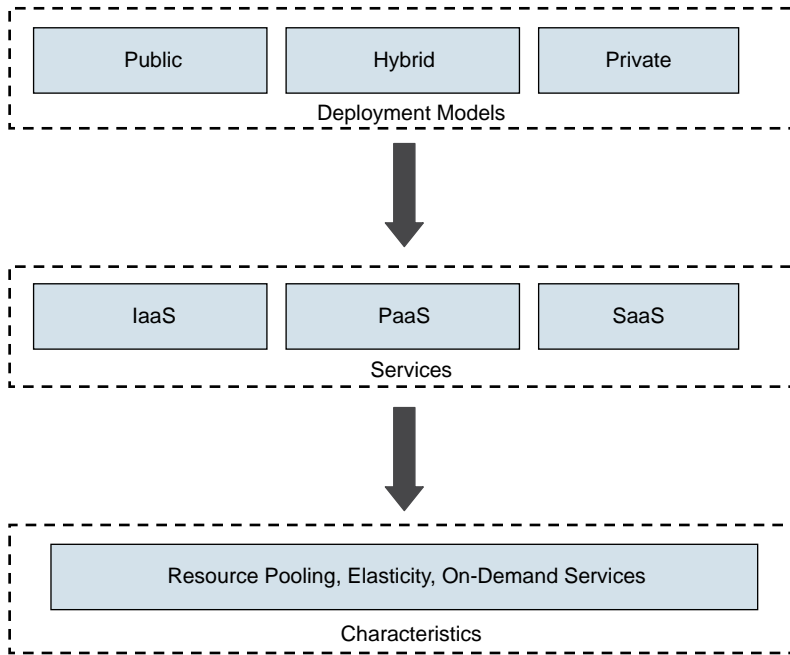


Figure 1-9 *Alignment of Deployment and Service Models*

Whether an organization leverages a public, private, or hybrid deployment model, the service offering defined for each environment and its characteristics will be aligned to general characteristics. Each deployment model can be leveraged to host the services defined based on elements considered for design.

Cloud Design Considerations

The following are the key essential features of cloud services:

- **On-demand service, pay-as-you-use**—Availability of the resource is based on a business condition.
- **Multitenancy and resource pooling**—Asset access is available from multiple locations across geographic boundaries.
- **Elasticity**—The user can scale storage, network bandwidth, and computing based on user demand.
- **Measured services**—The user can monitor flow based on consumer billing and has better management and reporting ability.

The cloud service model is completely different from a web-hosting environment, which usually leverages a hosting portal. For the deployment of these discussed features in the cloud, a stacked framework, such as the Cisco Domain 10 framework (see Figure 1-10), is necessary.

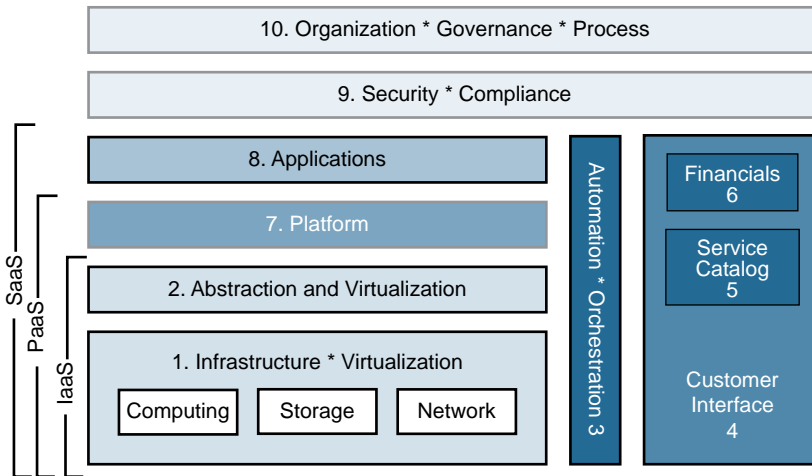


Figure 1-10—Cisco Domain 10 Framework

Cisco covers the stacked overview of cloud design in its Domain 10 framework. It is essential for you to understand the various domains to develop an effective cloud framework; the following sections provide details on the domains. Again, this is an example for understanding building blocks that must be considered for cloud capability. The Cisco Domain 10 framework provides a cloud architect focus area and details the dependencies between different blocks of the framework. Data center infrastructure for the cloud needs rapid provisioning service, and this requires an automation and orchestration framework to be aligned with computing, network, and storage domains. Good orchestration ties to the user catalog to speed up provisioning. The user catalog is a view for the user to provide available services and used services. The cloud has a pay-as-you-go model, so it is important for user catalog and finance to be integrated into a portal used for a customer interface. PaaS and SaaS are aligned to a more mature model of the cloud infrastructure. Security and governance is the top layer that needs to encompass all the other domains. The Domain 10 framework provides a sequential way to build a mature cloud model.

Domain 1: Infrastructure and Environmental

The first block of the Domain 10 framework is infrastructure and environmental, which is the base of any data center environment. The computing, storage, and network hardware elements and the supporting facilities, such as power and cooling, are part of this layer. The following are some of the characteristics that should be reviewed:

- Computing:
 - The platforms considered should have good management and should simplify deployment in the virtualized environment.
 - The platforms considered should be scalable and should be able to provide coverage for tenants that require increased performance, security, and visibility.
- Network:
 - The platforms considered should provide ease of deployment and be manageable.
 - The platforms considered should provide high-speed connectivity and simplify tenancy.
 - The platforms considered should provide ease of integration of security for the tenants.
 - The platforms considered should provide modular designs to improve availability and resiliency.
- Storage:
 - A rich, functional Fibre Channel SAN or IP-based mechanism should be available and capable of extending across multiple tenants.
 - The platforms considered should provide a single network with the flexibility to deploy Fibre Channel or IP-based protocols at any point in the path between server and storage.

Domain 2: Abstraction and Virtualization

A cloud-based design should involve a combination of abstraction and virtualization. The virtualization level should be available in networking, computing, and storage. The way that organizations think of computing resources has changed due to virtualization. Instead of managing an individual server, you can now manage virtual servers through a central management tool and concentrate more on new services than on the underlying server infrastructure. Storage virtualization integrates physical storage from multiple network storage devices to appear as one logical device. Network virtualization combines available network resources with computing elements and also allows tenancy aligned to security segments. While designing virtualization, the abstraction component hides the complexity of the virtualized elements and provides a simplified view of a data center's hardware, network, infrastructure, and storage resources as a single fabric.

Domain 3: Automation and Orchestration

Domain 3 depends on the selection of management automation software and how multiple domains are managed. A good automation process that lines up with the overall orchestration framework will replace several repetitive data center processes, such as provisioning a new server, thereby reducing the time line and cost to perform day-to-day

jobs. Workflow templates link multiple domains and help in provisioning new services at the data center.

Domain 4: Customer Interface

The end users utilize the Domain 4 portal for consuming new services. Users can select and use the services from the catalog menu. The customer interface should have an option for the user to select the services and get an overview of the governance. These two separate functionalities are integral to the customer portal.

The customer interface portal should link up with Domains 5 and 6, the service catalog and financials, so the users have a complete view of the services leveraged, the services available, and billing.

Domains 5 and 6: Service Catalog and Financials

Service catalog and financials encompass Domains 5 and 6 in the Domain 10 framework. Both of these domains must function closely with Domain 4, the customer interface domain. Customers can view the services by using the services catalog. The user has a list of orderable services supported by the IT department. The catalog also covers the service level agreement defined for each service. It is important to understand that the customer will receive greater operational efficiency and lower cost per unit when using the standard service catalog.

Financials enable an organization to have service-/usage-based billing. The integration of a chargeback model should cover all services and should be in alignment for the organization utilizing a hybrid cloud model. In a hybrid cloud model, the chargeback scheme for a public and on-premise private cloud environment should always be viewed and tracked.

Domains 7 and 8: Platform and Application

Domain 7 and 8 are platform and application blocks. As described in the previous sections, the platform portion provides an environment for developers with the required toolkit to develop applications.

The application portion is covered under SaaS. This model is commonly used for non-core enterprise-centric applications such as WebEx and sales force.

Domain 9: Security and Compliance

Security and compliance are top concerns for cloud adopters. The security design model should have the following criteria defined:

- **Segmentation**—You should enforce consistent policies and boundaries to protect data. This involves establishing boundaries for the network, computing domains, and policy-enforcement points for these domains.
- **Threat defense**—You should protect the asset resources and deploy efficient monitoring to detect attacks from internal or external sources.

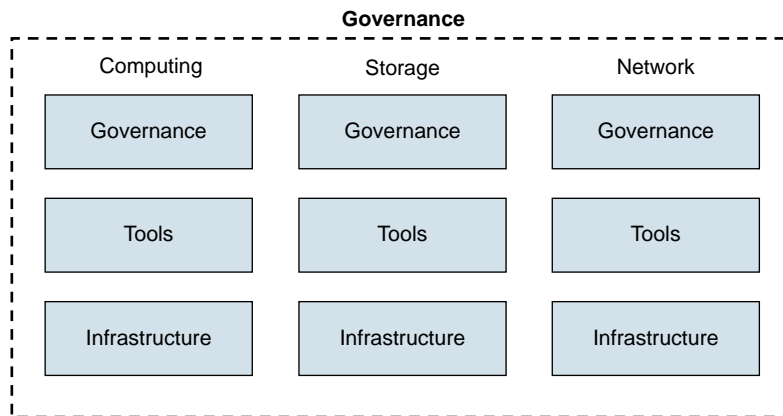
- **Visibility**—You should view each security domain and provide insight for compliance management. The key goal is to simplify operational visibility into various security zones and simplify compliance reporting.

Domain 10: Organization, Governance, and Process

Domain 10 influences the business objectives for the data center process. Aligning organization objectives with governance metrics results in efficient business, cost reduction, and improved user experience.

After cloud adoption, the governance process should be a single thread for computing, storage, and network as a converged infrastructure, as shown in Figure 1-11.

Traditional Data Center



New Model

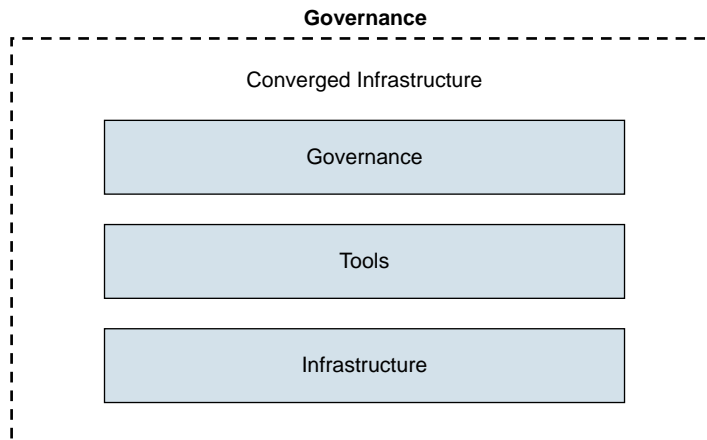


Figure 1-11 *Governance Process for Converged Infrastructure*

The traditional model of governance uses separate tools and processes for computing, storage, and network. These tools line up to a separate governance model that is confined to the team managing the respective domain. Governance needs two aspects: tools and teams responsible for the converged infrastructure. The team looking at and managing this data also must be collectively representing all the domains of the converged infrastructure. A core team that includes members and management representatives from each domain ties the combined governance. This model still leverages separate tracks of tools, processes, and governance. A data center user has a unified view of all the domains under a single service umbrella. This unified view is difficult to manage with the traditional model. Mature cloud governance leverages a consolidated tool set to view and manage all the domains.

Enterprise Connectivity to the Cloud

In any cloud design, it is essential to understand the enterprise connectivity to the off-premise cloud provider. The design solution and service level agreement for user access to the cloud depends on the access selected to connect to the cloud provider.

There are two main categories for connecting to the cloud provider:

- Internet for transport
- Direct connectivity to the cloud provider

The following sections describe these two categories.

Internet for Transport

The Internet for transport category involves using Internet access directly. Here, the user logs in to an application portal that is directly hosted by the provider. The user login credentials have authentication parameters. Secure transport via SSL capability can be used based on application requirements. The user utilization and access to the cloud are restricted to accounts based on the license agreement. This type of access is mostly used by organizations consuming via the SaaS model.

Accessing the Internet via a VPN has different options, including the following:

- **Hardware VPN access**—The customer accesses the cloud provider via VPN tunnel from the enterprise data center to the VPN edge gateway located at the cloud provider's network. After decryption at the VPN edge gateway, the user traffic traverses to the respective virtual data center (VDC) hosted in the cloud. In this model, the enterprise can reuse the existing hardware for VPN function and Internet connection to access the cloud provider. Bandwidth and throughput limitation needs to be factored here. Figure 1-12 shows the hardware VPN option.

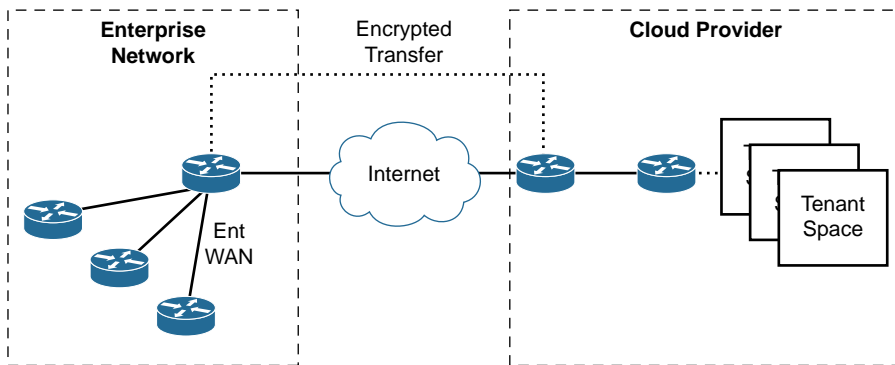


Figure 1-12 *Hardware VPN Option*

- **Software VPN access**—Software VPN access is popular for small groups of developers using the cloud infrastructure. To connect to the assets that are being hosted at the cloud provider, a VPN connectivity option can be leveraged. VPN high availability is the customer’s responsibility. As shown in Figure 1-13, the termination of a software VPN is inside the VDC or the tenant zone assigned for the entity.

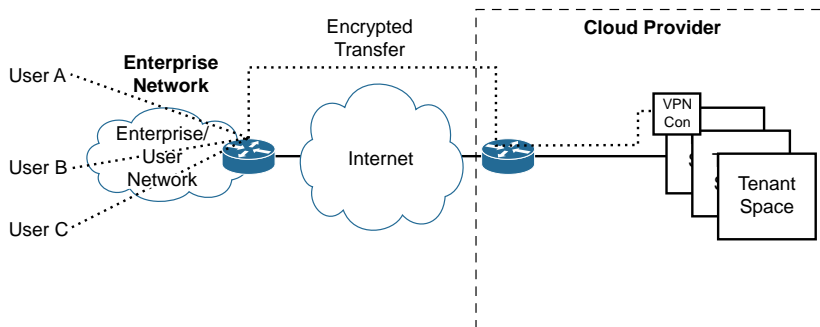


Figure 1-13 *Software VPN Option*

- **Hardware hub VPN access**—In this type of access, the organization has multiple connections to the cloud provider. The cloud provider provides a hub VPN service. Multiple locations within the enterprise space can be connected to the cloud provider. Here, the key feature is connectivity from multiple enterprise locations, which reduces the hair pinning of traffic from diverse geographic locations within the enterprise network. The term *hair pinning of traffic* is used when the packets traverse a transitory point between source and destination and is often seen when no direct path between the source and destination exists. The term is often used in WAN architectural designs. Figure 1-14 shows a high-level example of a hardware VPN hub.

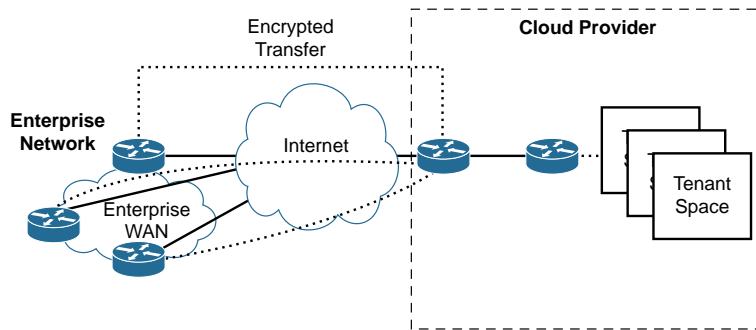


Figure 1-14 *Hardware Hub VPN Access*

Direct Connectivity to a Cloud Provider

With direct connectivity to a cloud provider, the customer traffic is not routed through the Internet; instead, the customer has a dedicated circuit to the cloud provider space. This dedicated circuit can be a leased link, such as an MPLS Layer 3 VPN or MPLS Layer 2 offering. It is recommended to use direct connectivity to meet strict SLA requirements for enterprise access applications hosted at the cloud provider space. Note that transport SLA-based access criteria is best suited to this option. Figure 1-15 shows a high-level example of a hardware VPN hub.

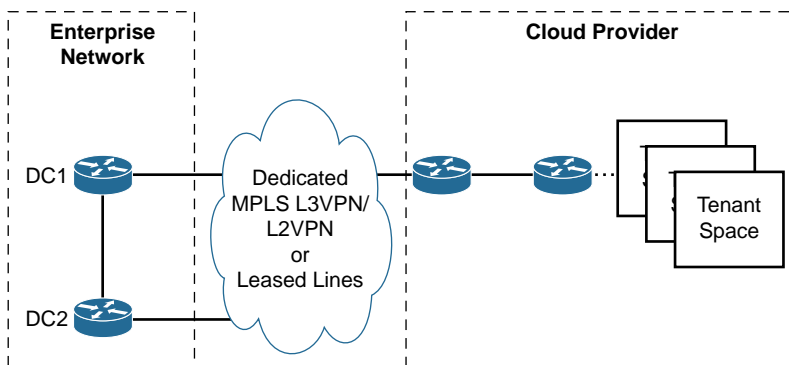


Figure 1-15 *Direct Connectivity to the Cloud Provider*

Future peer access options will evolve, with close partnerships between the service providers and cloud providers. The partnerships will enable end users to leverage geographic cloud provider locations to offload traffic from service providers to a cloud provider's localized data centers. This service will benefit large enterprises with global presence. The enterprise will get better transport SLA in the contractual agreement for application access through this access approach. The cost of maintaining a transport infrastructure will be offset to the cloud provider and service provider, thereby reducing the operating expense for the enterprise customer.

Enterprise Cloud Adoption Challenges

The following challenges need to be considered for cloud adoption:

- **Interoperability:**
 - In many cases, the enterprise application needs to implement changes before getting hosted in the cloud space. A common change such an application needs to adopt is re-IP addressing the application stack. This is common for organizations adopting the IaaS model.
 - Provider lock-in should be considered for an enterprise planning to adopt cloud services.
 - Traditionally practiced provisioning standards need to align with cloud provisioning standards.
- **Security and compliance:**
 - Security is a key factor for an enterprise to adapt to a suitable cloud model.
 - Regulatory factors dictate the adoption of a cloud model.
 - Every organization would like to be able to enforce and control enterprise security policies in the cloud space, without the management control of the cloud provider.
- **Visibility:**
 - Visibility is very important for enterprise admins to cater to the application service level agreement in the cloud.
 - It is also important to manage the operational aspect. The enterprise strives for operational visibility in transport and security spaces in the cloud environment. Some of these parameters need to be mentioned in the contractual agreement with the cloud provider.

The challenges of cloud adoption can be mitigated by a good cloud adoption strategy. These are the key elements of such a strategy:

- Scalability requirements
- Application profiling
- Use case application for cloud adoption
- Strategy to leverage cloud computing (like IaaS) versus cloud services (like SaaS)
- Governance and business service level objectives (SLO) for vendor selection and SLAs to be considered by the provider
- Metrics and governance roadmaps
- Visibility to the asset elements managing the enterprise tenant in the cloud

Enterprise architects prefer having control of the services in their tenant space within the cloud infrastructure. The concept of network function virtualization (NFV) comes up here. NFV elements are prevalent in IaaS cloud services. NFV brings a simple concept of implementing network service elements in software such as routing, load balancers, VPN services, WAN optimization, and firewalls. This is possible due to the new capability of provisioning memory and server facility to the network service elements. The provisioning of the network services is aligned with server elements. The NFV elements can be automated in the same workflow related to the application services. This enables faster provisioning of service with one orchestration device for network and application services. The data center design with virtual network services reduces the complexity of placing firewall or load balancing services because they are now closer to the asset. These virtual services enable an enterprise to launch these capabilities in an on-premises data center or in the provider cloud.

Software-Defined Networking

Most people understand that software-defined networking (SDN) is a new paradigm for networking that will foster agility and innovation in network services. However, trying to get a concise definition of SDN is like the old story about asking a group of blind men to describe an elephant when each has a different perspective of the animal. In the story, a group of six blind men encounter an elephant and try to learn what it is. Each of the men feels a different part of the elephant, and they all therefore describe the elephant a bit differently because they are influenced by their individual experiences:

- The first man approaches the elephant's side and believes it to be a huge wall.
- The second man, feeling the legs, believes they are trees trunks.
- The third man reaches out to the tusk and believes it to be a spear.
- The fourth man, who finds the elephant's trunk, believes it to be a snake.
- The fifth man, touching the ear, believes it to be a fan.
- The last man, who happens to seize the swinging tail, believes it to be a rope.

The moral of the story is that all the men are partly right, but all of them are also completely wrong. People can interpret SDN differently depending on their technology perspective. For a network administrator, SDN may mean automation and orchestration to simplify network operation. An architect may have a big-picture view of the network and focus on a controller-based protocol like OpenFlow. Here are some common interpretations of SDN:

- Network virtualization in the cloud
- Dynamic service chaining for instantiation of services
- Dynamic traffic engineering

- Simplified network orchestration and configuration
- Network function virtualization (NFV)

Open Networking Foundation

The Open Networking Foundation (ONF) is a nonprofit industry consortium focused on improving networking through SDN. The ONF defines SDN as follows:

The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices.

ONF focuses on the use of the OpenFlow protocol to drive the decoupling of network control and the forwarding function. OpenFlow is a standard-based communication interface between the controller and the forwarding layer of the SDN network. A controller defines a path that data packets traverse and programs the forwarding information into the network devices through OpenFlow application programming interfaces (API). Figure 1-16 shows a logical view of OpenFlow network programmability.

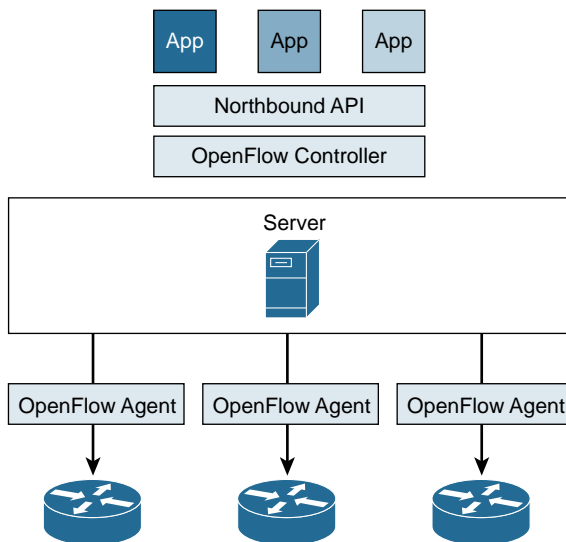


Figure 1-16 *OpenFlow Network Programmability*

OpenFlow began as a Stanford University PhD student's project call Ethane. Ethane was intended to centrally manage global policy using a flow-based network controller for network security. The idea led to what is now known as OpenFlow. Stanford University released OpenFlow version 1.0 as OpenFlow Switch Specification in 2009. It formed the basis for the subsequent releases of the protocol. ONF released OpenFlow version 1.2 in 2012. It was the first release of a specification by ONF after ONF formally

charted the project. Since then, ONF has released multiple versions of updates to the OpenFlow standard.

OpenFlow allows direct access to manipulate the forwarding plane of network infrastructure devices from different vendors, using a central controller. OpenFlow leverages the concept of flows to classify network traffic based on predefined rule sets that are generated either statically or dynamically through the SDN controller. A *flow* is a sequence of data packets traversing a network infrastructure that share a set of header attributes, such as the same source and destination IP addresses or the same protocol port identifier. The controller verifies that each flow is permitted by network policy before programming the flow entry in each device along the data path.

A network administrator can leverage OpenFlow to define how data traffic flows through a network infrastructure based on criteria such as bandwidth requirements, application usage patterns, and resource requirements. OpenFlow gives network operators very granular control while enabling the network to respond to real-time changes at the application and session levels.

Most of the time when SDN is discussed, OpenFlow is used as the interface to program the forwarding behavior of network devices. ONF views the OpenFlow protocol as an enabler for SDN and says it has the benefit of increasing interoperability of network equipment across a multivendor environment. However, there are alternatives to the use of OpenFlow for programming the network infrastructure, such as Network Configuration Protocol (NetConf) and Extensible Messaging and Presence Protocol (XMPP).

OpenDaylight Project

Organizations such as ONF are giving definitions for SDN that focus on the separation of the control plane and forwarding plane of the network, but the broader interpretation of SDN is that it's a framework that provides programmability to the network infrastructure. SDN enables IT agility, allowing network administrators and engineers to respond quickly to changing business requirements. Through SDN, application developers can leverage infrastructure as a platform to easily instantiate and integrate network services and applications through the use of APIs and scripting languages.

OpenDaylight is an open source project under the Linux Foundation Collaborative Projects. Industry leaders have formed OpenDaylight with the mutual goal of accelerating adoption and fostering innovation to the SDN framework. OpenDaylight can be a nucleus to any SDN architecture because it has a modular and extensible controller as its core. The OpenDaylight controller is implemented in software and is contained within its own Java virtual machine (JVM). This means the controller can be deployed across a wide variety of hardware and operating systems that support Java.

The OpenDaylight architecture has a modular southbound plugin framework for a multivendor environment. OpenDaylight offers a flexible and extensible interface for northbound communication, leveraging Java and RESTful APIs for multiple programming options to build applications to communicate with the controller.

OpenDaylight Hydrogen is the first release of the modular open source SDN platform. This release contains support for protocols such as OpenFlow 1.3, Open vSwitch Database Management Protocol (OVSDb), Border Gateway Protocol (BGP), and Path Computation Element Protocol (PCEP). As part of the Hydrogen release, OpenDaylight also contains a plugin for OpenStack Neutron. OpenDaylight uses northbound APIs to interact with OpenStack Neutron and uses OVSDb for southbound configuration of vSwitches on compute nodes.

Beryllium is the fourth software release for the OpenDaylight project. This release takes another step closer to creating an open industry platform for SDN and network function virtualization (NFV). OpenDaylight Beryllium provides a tighter integration to OpenStack and allows centralized network orchestration and management from the controller. Figure 1-17 shows the key standards supported by OpenDaylight Beryllium.

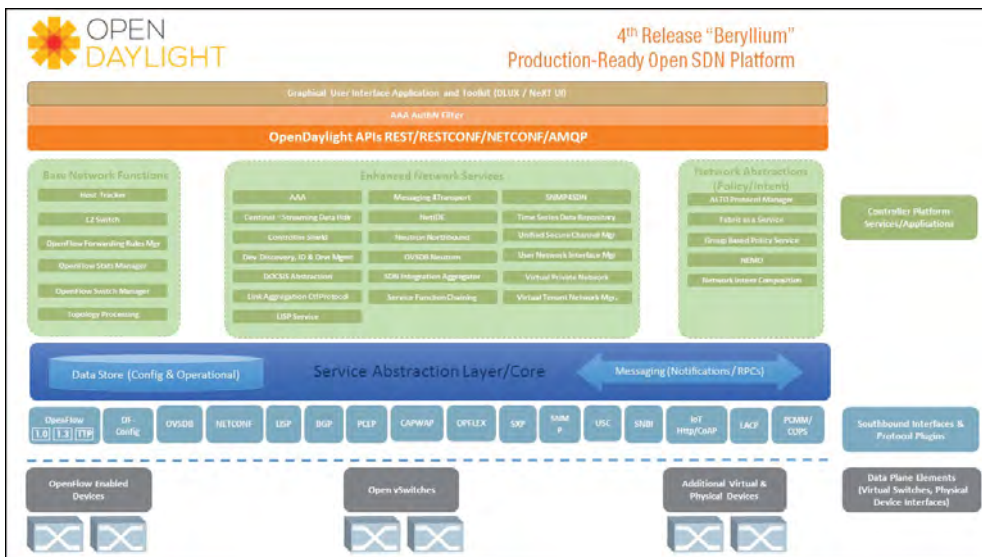


Figure 1-17 OpenDaylight Beryllium Framework

Network Function Virtualization

OpenDaylight provides a solid platform for network function virtualization (NFV). NFV offers new ways to design, orchestrate, and manage network services. NFV decouples network functions from underlying hardware so these functions can run as software images on common hardware as well as custom-built hardware. NFV is a framework that provides virtualization of network services such as routing, load balancing, firewall services, intrusion detection and prevention, and network address translation into building blocks. These services can be chained together to create network service chains tailored for different use cases.

The concept of NFV originated from service providers looking to increase the agility and flexibility of deploying new network services to support growing customer demands. NFV is complementary to SDN, and there is no dependency between SDN and NFV. NFV can be implemented using non-SDN mechanism leveraging techniques commonly deployed in many data centers. However, combining SDN with NFV simplifies deployment and operation and maintenance procedures.

OpenStack

OpenStack is an open source solution for building and managing open source cloud computing platforms that can be used in public and private clouds. OpenStack is managed by the OpenStack Foundation, a nonprofit organization overseeing both development and community building of the project.

The following are the major component services for OpenStack:

- **Nova**—Nova is the primary computing engine behind OpenStack. It is a controller for deploying and managing large numbers of virtual machines and handling computing tasks.
- **Glance**—Glance provides imaging service for OpenStack and enables images to be used as templates when deploying new virtual machines.
- **Neutron**—Neutron provides the networking capability for OpenStack. It ensures that the various components in an OpenStack deployment can communicate with one another effectively and efficiently.
- **Swift**—Swift is a storage system for objects and files that allows developers to refer to a unique identifier associated with a file or data instead of referring to files by their location on a disk drive. This allows better scaling and enables the system to manage how data is backed up.
- **Cinder**—Cinder is a block storage system that is analogous to traditional files storage where files are referenced by location on a disk drive. This function is necessary when data access speed is the most important consideration.
- **Horizon**—Horizon is the dashboard interface for OpenStack. It provides a graphical user interface for managing and orchestration to OpenStack for accessing all the components of OpenStack through APIs.
- **Keystone**—Keystone is the identity service for OpenStack. It consists of a central list of users mapped against all the components provided by OpenStack that the users have permission to access.

OpenStack enables users to deploy virtual machines and other component services previously described to handle various tasks for a cloud data center environment. OpenStack provides the infrastructure for users to quickly instantiate new services and allow developers to create software applications that tie into the framework for agile service delivery to the end users.

Summary

The intent of this chapter is to provide you with an understanding of the cloud environment and its key fundamental concepts. In it, you have learned about a number of concepts, including data center evolution, virtualization aspects for a data center, and different flavors of cloud services. It is important to understand these concepts to get a picture of the environment where a CSR 1000V is utilized. The following chapters get deeper into CSR 1000V.

This page intentionally left blank

Software Evolution of the CSR 1000

Now that you have reviewed the concepts of cloud and enterprise trends, this chapter provides an introduction to CSR 1000V. This chapter starts with some quick background on Cisco IOS that led to the development of the IOS XE architecture used in the ASR 1000 and ISR 4000 series platforms. This will help you better understand the CSR 1000V architecture.

IOS Software Architecture

The Internetworking Operating System (IOS) has been one of the primary operating systems for Cisco routers. IOS is a specialized operating system that runs on specialized hardware. It was first developed back in the 1980s to be a small embedded operating system for routers and switches. At the time, network devices had limited memory and CPU processing power. Compared to other types of operating systems, IOS was designed to be very lean and efficient, to stay within the constraints of routers' memory size and CPU performance. To maximize the router performance for forwarding data packets, IOS was written with minimal operational overhead, by trading the extra memory fault protection for maximum network performance. To take care of the overheads, extra processes were added to Cisco IOS routers.

IOS uses cooperative multitasking that offers a simple and efficient scheduling method. The IOS scheduler is responsible for managing and scheduling all processes in the multitasking environment. It employs priority run-to-completion scheduling, allowing each process a chance to run as long as it needs to run before releasing control back to the scheduler. Each process is a single thread and is assigned a priority value. The high-priority processes run before the lower-priority processes. The high-priority processes can jump to the head of the line for CPU runtime, but high-priority processes may not take CPU cycles away from running lower-priority processes. The IOS scheduler maintains four separate queues:

- **Critical**—Reserved for system processes such as the scheduler itself and memory management processes.
- **High**—Assigned to processes that require quick response time, such as transferring incoming packets from the network interface to memory.
- **Medium**—Assigned to most IOS user-level processes.
- **Low**—For all processes running in the background for periodic tasks, such as logging messages.

To reduce the impact of runaway processes hogging CPU runtime and refusing to relinquish control back to the scheduler, the IOS scheduler uses a watchdog timer that can forcefully interrupt and terminate rogue processes.

IOS has a small kernel for CPU scheduling and memory management. Unlike the kernel for traditional operating system cores that run in a protected CPU environment, the IOS kernel is a set of components that run in user mode on the CPU with access to full system resources. There is no special kernel mode for the IOS kernel. The IOS kernel schedules processes, provides memory resource management, handles hardware interrupts, maintains buffer resources, traps software exceptions, and manages other low-level services.

Figure 2-1 provides a conceptual view of the IOS architecture.

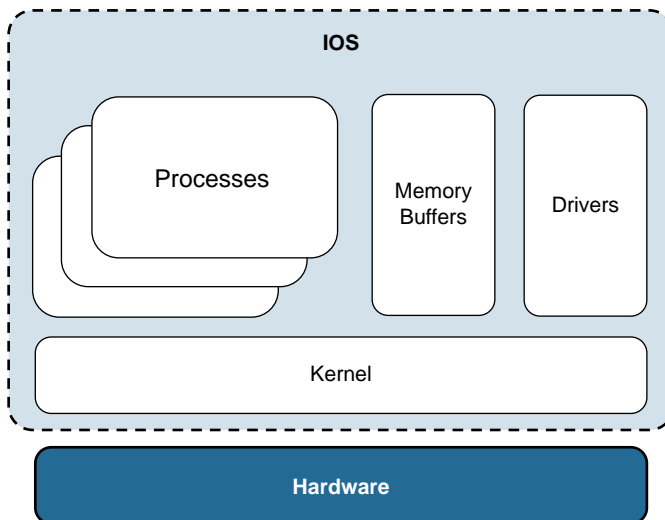


Figure 2-1 IOS Software Architecture

Over the decades, Cisco IOS has evolved into a feature-rich operating system powering Cisco hardware. Generic IOS has a 32-bit monolithic architecture that runs as a single image with all the processes having access to one flat memory address space; the kernel

does not support memory paging or swapping. The 32-bit architecture limits the memory allocation of IOS to 4GB (2^{32} bytes), and the single threading of IOS processes prevents the kernel from taking advantage of the multithreading capability in today's multicore CPU hardware architecture. Multithreading enables an operating system to execute multiple processes simultaneously. To keep up with the hardware advancements and to take advantage of the multithreading software capability on new multicore CPUs, Cisco developed IOS XE software.

IOS XE Architecture

IOS XE is part of the continuing evolution of the Cisco IOS operating system. IOS XE leverages key architectural components of IOS and at the same time overcomes the limitation of the 32-bit kernel of IOS. IOS XE retains the look and feel of classic IOS, while offering improved features and functionality. IOS XE separates system functions into the following components:

- IOS XE kernel
- IOS Daemon (IOSd)
- Forwarding Manager
- Interface Manager
- Platform Manager

Figure 2-2 shows a logical view of the IOS XE software architecture.

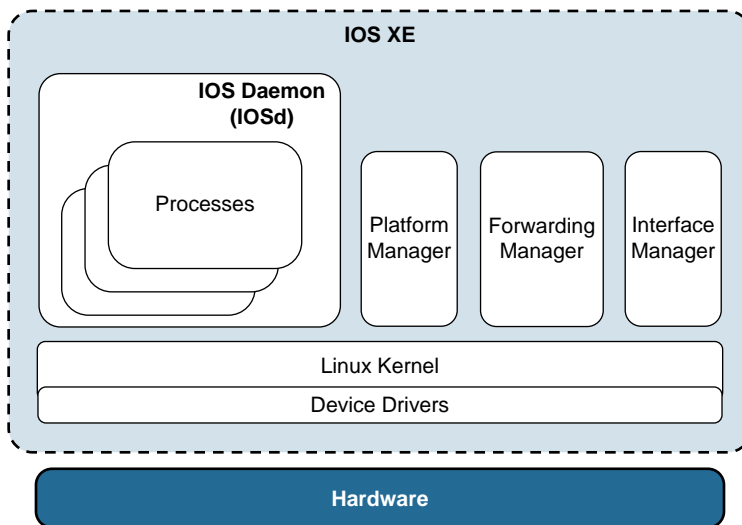


Figure 2-2 IOS XE Software Architecture

The IOS XE Kernel

IOS XE is based on a 64-bit hardened Linux kernel that offers significant enhancements over the classic IOS kernel. The Linux kernel architecture supports larger physical address space, allowing more than 4GB of memory to be addressed directly. The kernel offers multithreading support, providing the capability to execute multiple processes simultaneously across multicore CPUs. The operating system is built on a POSIX environment, which provides a set of system services targeted at the integration of network-aware applications.

The IOS XE Linux kernel, sometimes referred as BINOS in Cisco documents, leverages a scheduler to handle interrupt requests and share processor time among multiple processes. The IOS XE kernel offers a memory management system to manage process address space. The kernel resides in an elevated system state compared to regular user applications. It offers a protected memory management suite, which allows the kernel and other control plane applications to run in protected memory spaces. The IOS XE kernel uses symmetric multiprocessing (SMP) architecture, enabling applications to run across multiple CPU cores for higher performance. Unlike the classic IOS kernel, which features run-to-completion scheduling, IOS XE has a preemptive kernel that allows preemption of a task even as another task executes in the kernel. This capability provides better process response time and prevents a runaway process from hogging the CPU cycles. IOS XE leverages the Completely Fair Scheduler (CFS) to handle CPU resource allocation, with the goal of maximizing overall CPU utilization while also maximizing interactive performance.

The IOS Daemon

The IOS daemon (IOSd) is the classic IOS running as a user-level process scheduled by the IOS XE kernel. Being its own process allows IOSd to run in protected memory space, capable of restarting processes. IOSd is responsible for control plane processing, including network configuration, the command-line interface (CLI), routing protocol management, routing information base (RIB) computation, management of physical and logical interfaces, crypto IKE/IKEv2 negotiations, and processing protocols such as ICMP and SNMP. Communication with other components is done through messaging service. A shim layer included in IOSd enables it to communicate operations as messages to and from other components.

IOSd inherits a lot of characteristics from the classic IOS, and because it runs in a multithreaded 64-bit environment and is responsible for control plane functions, it enables the system to scale while maintaining backward feature compatibility. Internally, IOSd provides an environment controlled by its own process scheduler. IOSd runs in a protected address space that offers fault isolation from other components. IOSd uses the run-to-completion scheduler model for IOS control plane processes, but the process IOSd itself can be preempted by the Linux kernel scheduler. The Linux kernel leverages a CFS low-latency scheduler and preemptable threads to minimize scheduling delays.

IOS XE improves the overall security and stability of a system through several techniques. First, IOSd runs as a user-level process with root permission that has access only to its own memory and a restricted portion of the file system for enhanced fault containment. Second, to prevent overloading the IOS daemon, the information sent to IOSd is filtered and rate limited. Finally, some of the functions that were handled natively in classic IOS are offloaded to other components. Together, these methods reduce the likelihood of IOSd becoming overloaded and unable to process critical control plane packets.

The Forwarding Manager

The Forwarding Manager is responsible for propagating IOSd control plane operations to the forwarding data plane. It provides a bidirectional communication path between the IOSd and the data plane through an API setup. The Forwarding Manager programs the data plane and maintains the forwarding state of the system and is one of many abstraction layers that allow IOS XE to run on a variety of underlying hardware.

The Interface Manager

The Interface Manager is responsible for communicating IOSd events associated with the creation and bring-down of interfaces, both physical and logical, to the appropriate hardware I/O interface. It provides a communication channel for collecting and sending hardware interface statistics to IOSd.

The Platform Manager

The Platform Manager is in charge of the basic operation of the hardware platform, including the initialization and booting of the various processes for IOS XE. The Platform Manager monitors online insertion and removal (OIR) of interface card components and generates OIR notifications to IOSd. In addition, the Platform Manager provides a mechanism for periodically monitoring and storing critical data for hardware devices into nonvolatile memory. This includes board uptime and monitoring of environment data such as temperature and voltage.

Together, the Forwarding Manager, the Interface Manager, and the Platform Manager form the middleware that interconnects the major components. These three managers maintain the state of the overall system in nonvolatile memory and offer the capability of In-Service Software Upgrade (ISSU) on certain platforms.

Cisco ASR 1000 System Architecture Overview

The Cisco Aggregation Service Router (ASR) 1000 series router is a wide area network (WAN) and Internet edge routing platform. It was also the first platform for the Cisco IOS XE network operating system. The ASR 1000 router delivers embedded hardware acceleration for multiple IOS XE software services without the need for separate service

modules. It is also the foundation for the virtual routing platform CSR 1000V's system architecture.

Now let's look into the ASR 1000's high-level system architecture before examining the CSR 1000V. The ASR 1000 runs on IOS XE and has three main components: route processor (RP), embedded service processor (ESP), and SPA interface processor (SIP).

Route Processor

A route processor (RP) includes a general-purpose CPU and runs the IOS XE network operating system. It is an integrated system component for fixed chassis platforms such as the ASR 1001-X, and it is a separate module on modular systems such as the ASR 1004 and ASR 1006-X.

An RP handles all the control plane traffic and manages the system functions. It runs routing protocols, builds and distributes forwarding information to the ESP, negotiates and exchanges encryption keys in IPsec sessions, and monitors and manages power and temperature for system components including line cards, power supplies, and fans.

An RP is responsible for chassis management, including initialization of the ESP; network interfaces such as shared port adapters (SPA), SPA interface processors (SIP), or network interface modules (NIM); IOS XE image management; logging facilities; and distribution of user configuration to the ESP and SIPs.

Embedded Service Processor

An embedded service processor (ESP) is a centralized forwarding engine for a system. ESPs are responsible for the data plane processing tasks, and all network traffic flows through them. The ESP module performs all baseline router packet operations, including the following:

- MAC classification
- Layer 2 and Layer 3 forwarding of data packets
- Quality-of-service (QoS) classification, policing, and shaping
- Security access control lists (ACLs)
- VPNs
- Load balancing
- NetFlow statistic reporting
- Firewall
- Network-based application recognition
- Network address translation (NAT)
- Hardware-assisted encryption

At the heart of an ESP is the Cisco QuantumFlow Processor (QFP). The QFP is a multicore parallel packet processor on a single silicon chip, designed for scaling and performance while offering rich data path features and services. The QFP architecture is non-pipelined, parallel processing with centralized shared memory. The QFP packet processor engine is responsible for processing all traffic flows in the data-forwarding path. Inside, the QFP also includes a traffic manager responsible for queuing and scheduling functions for the forwarding plane. An ESP performs all packet forwarding, buffering, and output queuing and scheduling for all traffic going through the QFP.

SPA Interface Processor

The SPA interface processor (SIP) is a carrier card that provides the physical and electrical connectivity for the shared port adapters (SPA). It offers two levels of packet prioritization for ingress packets from the SPAs and a large buffer for queuing ingress packets going to an ESP for processing. A SIP has a smaller egress buffer to prevent output interfaces from overrun by the ESP scheduler and ensures full link utilization in the transmit direction. A SIP is also responsible for generating an egress queueing event to the ESP, providing a back-pressure mechanism for engaging the traffic manager for egress packet shaping.

The three components just described have powerful control processors dedicated for control and management functions. Figure 2-3 illustrates these three system components of the ASR 1000 hardware architecture.

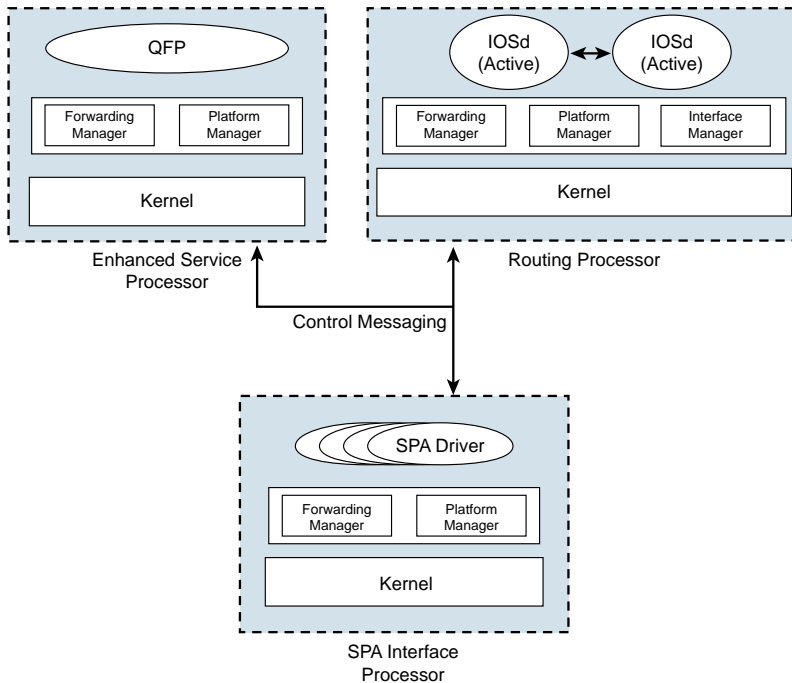


Figure 2-3 Conceptual View of ASR 1000 Architecture

Cloud Service Router 1000V Overview

Enterprises are increasingly virtualizing their applications and services for cost savings and leveraging services (that is, computing cycles and storages) hosted by cloud providers for more agility and flexibility. The need to securely move applications from a private data center to the cloud to take advantage of the benefits of virtualization and cloud computing led to development of the CSR 1000V.

The CSR 1000V is an IOS XE software router based on ASR 1001 that runs within a virtual machine deployed on any general server hardware or x86 server hardware. There are a lot of commonalities between the system architecture for the CSR 1000V and the ASR 1000; however, there are some differences as well. Let's take a look at how the CSR 1000V, which works on the IOS XE software, is different from the ASR 1000:

- The CSR 1000V is a virtual router that does not have SPA components.
- The CSR 1000V does not have the hardware interconnects, SPAs, and the few kernel utilities that relate to SPA.
- Utilities have been added to the kernel for use by virtual hardware presented by the virtualization layer, such as vCPU, vMemory, and vConsole.
- There is no crypto ASIC. The CSR 1000V leverages AES-NI (with compiled instructions into the crypto library). However, in the future, as chip technology advances, crypto support can be leveraged from the hardware.
- Due to the absence of the QFP, it has lower forwarding performance compared to a traditional ASR 1000.

Note The CSR 1000V can have good performance values with appropriate hardware and hypervisor tweaking. You learn more about this in Chapter 3, “Hypervisor Considerations for the CSR.”

Figure 2-4 provides a high-level overview of the CSR 1000V.

The Cisco CSR 1000V series lowers the barriers to enterprise adoption for cloud. The primary features include the following:

- Flexible virtual form factor designed for multitenant, provider-hosted clouds
- Complete hypervisor-isolated, multiservice router instance for each tenant
- Proven, familiar, enterprise-class Cisco IOS Software networking services
- Feature and operational consistency with Cisco physical form-factor routers
- Component of end-to-end WAN architecture with Cisco Integrated Services Routers and Cisco Aggregation Services Routers

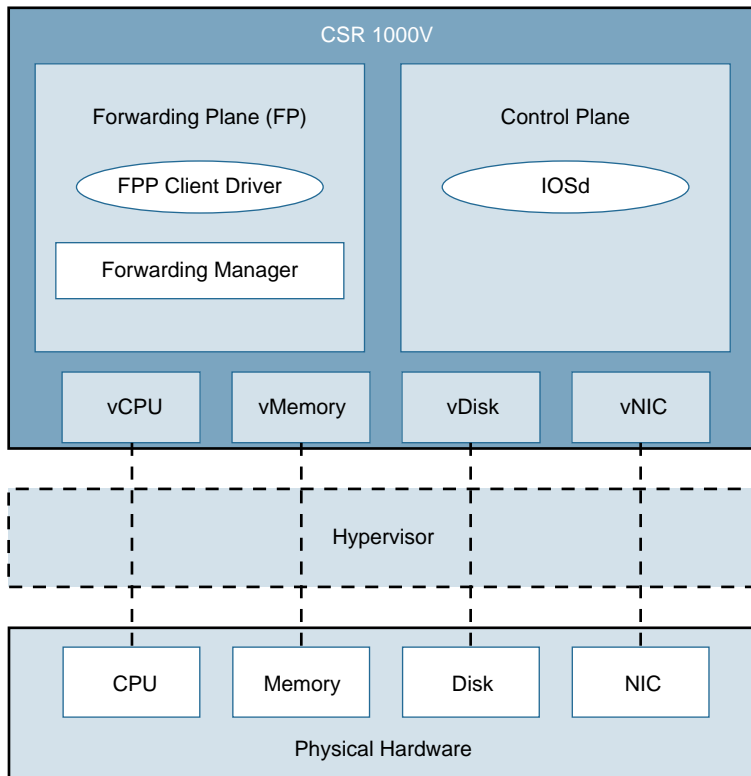


Figure 2-4 Overview of the CSR 1000V

Chapter 4, “CSR 1000V Software Architecture,” provides detailed coverage of the software architecture for the CSR 1000V.

Deployment Requirements

Deployment of the CSR 1000V requires the following for an infrastructure-agnostic feature set:

- Cisco UCS server and any x86 servers
- vSwitch, OVS, dVS, N1KV (note that there is no dependency on CSR 1000V and vSwitch)
- VMware ESXi 5.0 or above, Kernel-based Virtual Machine (KVM), RHEL 6.6, Ubuntu 14.04 LTS Server, Microsoft Hyper-V, Windows Server 2012 R2, Citrix XenServer 6.2, Amazon Web Services (AWS)

The CSR 1000V has the following hardware requirements for deployments:

- Minimum of one vCPU with scalable up to two, four, and eight vCPU for performance. This requirement will change in future releases. Please refer to the release notes for minimum requirements on the latest release.
- 2.5GB on one vCPU (default) and 4GB on four vCPUs. Memory elasticity is supported with an 8GB license (expansion from 4GB to 8GB).
- 8GB hard drive with local, SAN, and NAS resources supported.

Table 2-1 lists the support for various hypervisors by CSR 1000V.

Table 2-1 *Summary of CSR 1000V Hypervisor Support*

| CSR Release | VMware ESXi | Citrix XenServer | Microsoft Hyper-V | KVM |
|--------------------|---------------------------|-------------------------|--------------------------|---|
| 3.9S | 5.0 | Not supported | Not supported | Not supported |
| 3.10S | 5.0 5.1 | 6.0.2 | Not supported | Linux KVM based on Red Hat Enterprise Linux 6.31 Red Hat Enterprise Virtualization 3.1 |
| 3.11S | 5.0 5.1 | 6.0.2 | Not supported | Linux KVM based on Red Hat Enterprise Linux 6.31 Red Hat Enterprise Virtualization 3.1 Ubuntu 12.04.03 LTS Server 64 Bits |
| 3.12S and 3.13S | 5.0 5.1 5.5 | 6.1 and 6.2 | Windows Server 2012 R2 | Linux KVM based on Red Hat Enterprise Linux 6.31 Ubuntu 12.04.03 LTS Server, 64 bits |
| 3.14S | 5.0 5.1 5.5 | 6.2 | Windows Server 2012 R2 | Linux KVM based on Red Hat Enterprise Linux 6.5 Ubuntu 14.04 LTS Server, 64 bits |
| 3.15S and 3.16S | 5.0 5.1 5.5 6.0* | 6.2 | Windows Server 2012 R2 | Linux KVM based on Red Hat Enterprise Linux 6.6 Ubuntu 14.04 LTS Server, 64 bits |

| CSR Release | VMware ESXi | Citrix XenServer | Microsoft Hyper-V | KVM |
|-------------|-------------|------------------|------------------------|---|
| 3.17S | 5.0 | 6.2 | Windows Server 2012 R2 | Linux KVM based on Red Hat Enterprise Linux 7.1 |
| | 5.1 | | | |
| | 5.5 | | | Ubuntu 14.04 LTS Server, 64 bits |
| | 6.0 | | | |

* VMware ESXi 6.0 supported on Cisco IOS XE 3.16.1S and later, and 3.17S and later.

The support for newer versions of the hypervisor will be based on software revisions for the CSR 1000V. Please refer to the following CCO link to verify support of the hypervisor on the latest CSR 1000V release: <http://www.cisco.com/c/en/us/td/docs/routers/csr1000/software/configuration/csr1000Vswcfg/csroverview.html#pgfId-1081607>. Chapter 3 describes the CSR 1000V's interaction with different types of hypervisors.

Elastic Performance and Scaling

The CSR 1000V is licensed based on performance and feature set and can be updated on the fly to adjust for increased performance requirements. Initially, when the CSR 1000V first boots up, the router is in evaluation mode. For IOS XE 3.12 and earlier releases, the network interfaces are activated, but the throughput is limited to 2.5Mbps and the feature support is restricted until the evaluation license is activated. In IOS XE 3.13 and later releases, the CSR 1000V boots with maximum throughput limited to 100Kbps but with the AX feature set enabled by default. The evaluation license can be enabled with the AX feature set with up to 50Mbps of maximum throughput. Alternatively, the evaluation APPX feature set can be enabled with 10Gbps maximum throughput.

There are several throughput options, ranging from 10Mbps all the way up to 10Gbps. The CSR 1000V applies a license shaper that restricts the aggregate throughput of the router's interfaces. For example, if a 100Mbps license is installed, a maximum of 50Mbps of bidirectional traffic is allowed through the CSR 1000V system.

The CSR 1000V uses a license-based shaper to enforce the throughput on the router. The licensed bandwidth is measured against the aggregate throughput of the traffic across all the interfaces on the router. The license shaper regulates both priority traffic and non-priority traffic. The shaper is implemented at the root of the QoS hierarchy. To ensure that the license shaper doesn't drop high-priority traffic, QoS (for example, LLQ) should be configured. All traffic exceeding the bandwidth licensed for the shaper will be tail-dropped.

The following shows the IOS XE release license shaper behavior:

- Cisco IOS XE 3.10S and earlier regulate throughput only on the non-management interface. The traffic going across the GigabitEthernet 0 management interface does not count toward the aggregate bandwidth for the system.
- Cisco IOS XE 3.11S and later enforce the license shaper across all interfaces, including the GigabitEthernet 0 management interface.

The license throughput shaper enforces globally and not on a per-interface basis. This means the license shaper does not distinguish the different types of traffic going across the system. When the aggregate bandwidth exceeds the licensed throughput, the excess packets are discarded. Figure 2-5 shows the three interfaces on the CSR 1000V passing an aggregate throughput of 120Mbps. This exceeds the 100Mbps licensed throughput, which means 20Mbps of traffic is discarded.

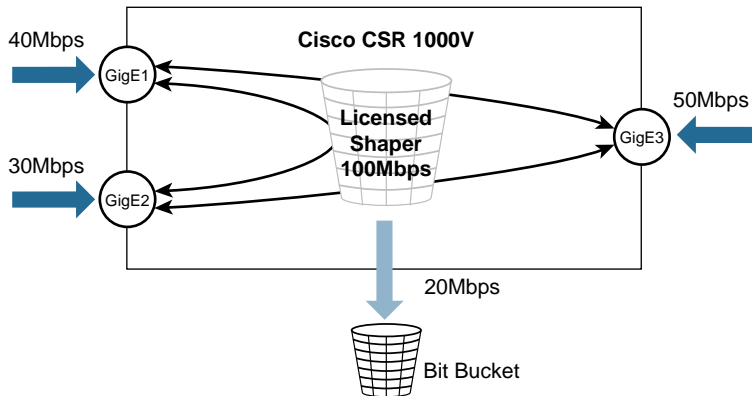


Figure 2-5 Conceptual Example of the CSR 1000V License Shaper

Table 2-2 shows the server resource requirement for the CSR 1000V for the different technology packages. Refer to the Cisco.com website for the features covered under the different technology packages.

Table 2-2 *CSR 1000V Resource Requirements for Each Technology Package*

| Throughput | Technology Package | | | |
|------------|--------------------|-------------|-------------|-------------|
| | IP Base | Security | APPX | AX |
| 10Mbps | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB |
| 50Mbps | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB |
| 100Mbps | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB |
| 250Mbps | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB |
| 500Mbps | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB |
| 1Gbps | 1 vCPU/4GB | 1 vCPU/4GB | 1 vCPU/4GB | 2 vCPUs/4GB |
| 2.5Gbps | 1 vCPU/4GB | 1 vCPU/4GB | 4 vCPUs/4GB | 4 vCPUs/4GB |
| 5Gbps | 1 vCPU/4GB | 2 vCPUs/4GB | 8 vCPUs/4GB | —* |
| 10Gbps | 2 vCPUs/4GB | —* | —* | —* |

* This performance and feature combination is currently not supported as of the IOS XE 3.16S release.

IP Base, Security, APPX, and AX are the technology packages described in Table 2-3.

Rapid Deployment and Routing Flexibility in the Cloud

The CSR 1000V is a software router that can be deployed as a virtual machine in a provider-hosted cloud environment or in an enterprise-owned virtual environment. In a cloud-hosting network, each tenant gets its own CSR 1000V instance that provides its own VPN connectivities, security, access control rules, QoS policies, and so on. In addition, the router can be deployed as a multitenant router, using virtual routing and forwarding (VRF), to maintain routing separation and feature configuration for each tenant.

The CSR 1000V's features are tied to the technology package installed on the router, based on the license purchased. Table 2-3 shows the four technology packages available and the features of each one. It is a best practice to refer to the Cisco Feature Navigator at <http://tools.cisco.com/ITDIT/CFN/jsp/index.jsp> to verify the latest feature support.

Table 2-3 *Features of the Technology Packages*

| Technology Package | CSR 1000V Features (Software 3.17 Release) |
|---|--|
| IP Base (Routing) | <p>Basic networking: BGP, OSPF, EIGRP, RIP, ISIS, IPv6, GRE, VRF-LITE, NTP, QoS</p> <p>Multicast: IGMP, PIM</p> <p>High availability: HSRP, VRRP, GLBP</p> <p>Addressing: 802.1Q VLAN, EVC, NAT, DHCP, DNS</p> <p>Basic security: ACL, AAA, RADIUS, TACACS+</p> <p>Management: IOS XE CLI, SSH, Flexible NetFlow, SNMP, EEM, NETCONF</p> |
| Security (Routing + Security) | <p>IP Base features plus the following:</p> <p>Advanced security: ZBFW, IPsec VPN, EZVPN, DMVPN, FlexVPN, SSLVPN</p> |
| APPX/APP | <p>IP Base features plus the following:</p> <p>Advanced networking: L2TPv3, BFD, MPLS, VRF, VXLAN</p> <p>Application experience: WCCPv2, APPXNAV, NBAR2, AVC, IP SLA</p> <p>Hybrid cloud connectivity: LISP, OTV, VPLS, EoMPLS</p> <p>Subscriber management: PTA, LNS, ISG</p> |
| AX (Routing + Security + APPX + Hybrid Cloud) | <p>Security features plus the following:</p> <p>Advanced networking: L2TPv3, BFD, MPLS, VRF, VXLAN</p> <p>Application experience: WCCPv2, APPNAV, NBAR2, AVC, IP SLA</p> <p>Hybrid cloud connectivity: LISP, OTV, VPLS, EoMPLS</p> <p>Subscriber management: PTA, LNS, ISG</p> |

CSR 1000V Deployment Examples

This section shows some deployment use cases for the CSR 1000V. Chapter 5, “CSR 1000V Deployment Scenarios,” and Chapter 6, “CSR Cloud Deployment Scenarios,” go into more detail about these use cases.

Secure Cloud VPN Gateway

A large enterprise typically has a central headquarters and a few regional hubs, two or more data centers, and hundreds (or even thousands) of branch offices. An enterprise organization needs to secure its connectivity between its private data center and its off-premise data center in the cloud.

Many public cloud and virtual private cloud services also provide VPN as a capability; however, this service is typically offered as a black box with little visibility and troubleshooting capability into the VPN service. In addition, there is a monthly recurring charge or per-VPN tunnel fee.

In this case, an enterprise can leverage the CSR 1000V in the cloud for secure connectivity while maintaining a consistent WAN architecture. Using the CSR 1000V as the VPN gateway in the cloud provides a familiar platform for monitoring and troubleshooting problems while avoiding any additional VPN service fees. The CSR 1000V can be used in a hub-and-spoke, partial-mesh, or full-mesh network. By leveraging DMVPN, an enterprise can dynamically connect its branch sites to its data center in the cloud, thereby minimizing the latency caused by backhaul through the central site while overcoming the complexity of managing point-to-point IPsec VPNs.

The following features are leveraged for a secure cloud VPN gateway:

- **VPN**—IPsec VPN, DMVPN, and FlexVPN
- **Routing protocol**—BGP, OSPF, and EIGRP
- **Security**—Zone-based firewall (ZBFW), access control list (ACL), and NAT
- **Host connectivity**—DHCP

Network Extension from Premises to Cloud

An enterprise that wants to maintain IP address consistency when moving an application from its private data center to an off-premise cloud environment can leverage the CSR 1000V to provide Layer 2 or Layer 3 network extensions. The CSR 1000V offers features such as Locator/ID Separation Protocol (LISP) to provide IP mobility and allow virtual machines to move from the enterprise's data center to the cloud for resource elasticity.

The Overlay Transport Virtualization (OTV) and Virtual Private LAN Services (VPLS) features on the CSR 1000V enable an enterprise to extend Layer 2 network and VLAN segmentation from its data center into the cloud for virtual machine migration. This provides the enterprise the capability for server backup, disaster recovery, and computing on-demand scaling.

The following features are leveraged for network extension from premises to cloud:

- LISP
- NAT
- OTV
- VPLS

Segmentation Within a Cloud

A service provider can offer managed cloud connectivity to its customers and provide performance and reliability guarantees by leveraging the CSR 1000V as customer edge routers. This allows the service provider to extend segmentation and to provide end-to-end connectivity into the cloud right up to the edge of the customer's segments within the cloud.

The CSR 1000V can participate in a Virtual Extensible LAN (VXLAN) network service as a VXLAN tunnel endpoint (VTEP) and provide a termination point for VXLAN network identifiers (VNI). For larger enterprise data centers and service provider networks, this feature overcomes the scaling limitation of 4096 VLANs for increased network scalability. A VXLAN supports millions of network identifiers and allows a service provider to support a greatly increased number of tenants on its existing infrastructure. An enterprise can also deploy the CSR 1000V as a dedicated VXLAN gateway to allow traffic to be routed or bridged to other VXLAN or non-VXLAN networks.

The following features are leveraged for segmentation within a cloud:

- MPLS VPN
- BGP
- VRF
- VXLAN

CSR 1000V Key Features

A number of key features of the CSR 1000V are commonly used in the cloud. The following is an alphabetical list of some of the most important of these key features:

- **Application Visibility and Control (AVC)**—Cisco AVC is a solution that enables application awareness in the network. AVC incorporates into the CSR 1000V application recognition functions with performance monitoring capabilities that were traditionally available only on hardware routers and switches. This integrated approach greatly reduces the network footprint, simplifies network operations, and reduces total cost of ownership.

The AVC solution leverages multiple technologies to recognize, analyze, and control more than 1000 applications, including voice and video, email, file sharing, gaming, peer-to-peer (P2P), and cloud-based applications. There are four functional components of AVC:

- **Application Recognition via Next-Generation Network Based Application Recognition (NBAR2)**—NBAR2 offers an innovative Deep Packet Inspection (DPI) technology for more than 1000 applications within the traffic flow. To address the evolving nature of applications, NBAR2's application signatures can be updated through a protocol pack while the router is in service. It is re-architected

based on the Service Control Engine (SCE) with advanced classification techniques to improve accuracy and increase signatures. Its application recognition engine supports more than 1000 applications and subclassifications. NBAR2 provides a field extraction mechanism to export predefined fields from packet headers via Flexible NetFlow (FNF) for reporting. With the NBAR2 protocol pack, new and updated application signatures can be loaded into routers without the need to upgrade the system OS. New signatures and signature updates are released monthly via protocol packs. NBAR2 is also capable of defining customized applications based on ports, payload values, or URL.

- **Performance Collection and Exporting**—AVC utilizes an embedded monitoring agent to collect application statistics, application response time (ART) metrics such as transaction time and latency for TCP applications, and packet loss and jitter information for voice and video applications. These metrics are aggregated and exported using a standard flow export format such as NetFlow version 9 and IPFIX.
- **Management Tool**—With open flow export formats such as NetFlow version 9 and IPFIX data export, Cisco Prime Infrastructure and other third-party network management tools can consume data exported by AVC for application and network performance reporting. This gives users flexibility to utilize Cisco management tools or to leverage other management tools of their choice.
- **Control**—By utilizing NBAR2, AVC devices can reprioritize critical applications or enforce application bandwidth use using industry-leading Cisco QoS capabilities. In addition, the routers can provide intelligent application path selection based on real-time performance with Cisco Performance Routing (PfR).
- **Dynamic Multipoint VPN (DMVPN)** —DMVPN is an embedded security feature on the CSR 1000V for building scalable IPsec VPNs that support distributed applications. DMVPN is widely used to combine enterprise branch, teleworker, cloud network, and extranet connectivity. DMVPN offers the capability to allow branch offices to communicate directly with each other over the public WAN or Internet by building dynamic, on-demand VPN connections between sites. DMVPN enables zero-touch deployment of IPsec VPNs and improves network performance by reducing latency and jitter, while at the same time optimizing bandwidth usage at the head end locations.

DMVPN offers several major benefits:

- Lowers capital expenditures (CapEX) and operating expenses (OpEx) and reduces deployment complexity by providing zero-touch deployment.
- Simplifies branch office communication by enabling dynamic branch-to-branch secure connectivity such as voice and video services.
- Improves business resiliency and data integrity and security by incorporating routing with Advanced Encryption Standard (AES).

- **Embedded Event Manager (EEM)**—IOS EEM is a powerful and flexible subsystem running on the CSR 1000V that provides real-time network event detection and onboard automation. It enables you to adapt the behavior of your network devices to align with business needs.

EEM supports more than 20 event detectors that are highly integrated with different Cisco IOS software components to trigger actions in response to network events. You can inject your business logic into network operations using IOS EEM policies to enable creative solutions, such as automated troubleshooting, fault detection, and device configuration.

- **IP Service Level Agreement (IP SLA)**—The Cisco 1000V offers embedded IP SLA capability, allowing customers to understand IP service levels, increase productivity, and reduce operational costs in the cloud. The IP SLA feature performs active monitoring of network performance and can be used for network troubleshooting, readiness assessment, and health monitoring. IP SLA running on the CSR 1000V can be configured to actively monitor and measure performance between multiple network locations or across multiple network paths. The IP SLA sends out probes that can simulate network data, voice, or video services and collect network performance information in real time. The information collected includes data about response time, one-way latency, variation in packet delivery (jitter), packet loss, voice quality scoring (MOS score), network resource availability, application performance, and server response time. You can use the measurement and statistics provided by IP SLA for troubleshooting, problem analysis, and network capacity planning.
- **Location/ID Separation Protocol (LISP)**—LISP is a routing architecture that offers new semantics for IP addressing. The current IP routing and addressing scheme uses a single numbering space, the IP address, to express two pieces of information:
 - Device identity
 - How the device attaches to the network

The LISP routing architecture provides separation of the device identity from its location. This capability brings enhanced scalability and flexibility to the network, enabling virtual machine IP mobility (VM-Mobility) for geographic dispersion of data centers and disaster recovery. In addition, LISP simplifies enterprise multihoming with ingress traffic engineering capability, multitenancy over Internet, and simplified IPv6 transition support.

The LISP VM-Mobility solution addresses the challenge of route optimization when a virtual machine moves from one location to another. It does this by keeping the server identity (its IP address) the same across moves so the clients can continue to send traffic regardless of the server's location, and at the same time, it guarantees optimal routing between clients and the server that moved.

- **Multiprotocol Label Switching (MPLS) VPN** —MPLS is a packet-forwarding technology that uses labels to expedite data packet forwarding. A label is a short 4-byte identifier inserted between the Layer 2 header and Layer 3 header of a data

packet. One key benefit of MPLS is that the decision about where the packets are forwarded is based solely on the label and not on the Layer 3 information the packet carries; this allows for faster lookups for the forwarding decision.

MPLS VPN extends the capabilities of MPLS and supports creation of VPNs across an MPLS network. MPLS may be used to deliver VPN solutions at either a Layer 2 VPN (L2VPN) or Layer 3 VPN (L3VPN). All solutions enable a service provider to deliver a private service over a shared network infrastructure.

An MPLS L3VPN provides a full-mesh Layer 3 virtual WAN service to interconnect customer edge (CE) routers. The segmentation of traffic between the customers is done through the use of Virtual Routing and Forwarding (VRF) and MPLS VPN labels for traffic separation.

An MPLS L2VPN offers “switch in the cloud” forwarding service. L2VPN provides the capability to extend Layer 2 connectivities between sites and the ability to span VLANs across sites.

- **Performance Routing (PfR)** —PfR is a Cisco innovation that delivers intelligent path control for application-aware routing across the WAN. PfRv3 is the third generation of this intelligent path control capability. It offers simple, centralized configuration, improved application monitoring, and faster convergence. PfRv3 monitors application performance on a per-flow basis and applies the statistics collected to select the best path for that application. PfR provides the capability for dynamic selection of the best path for application-based business policies, as well as application-based load balancing across paths for full utilization of bandwidth.
- **Virtual Private LAN Services (VPLS)** —VPLS is a Layer 2 VPN service that provides multipoint Ethernet LAN services to multiple sites, offering a single bridged domain over a managed IP or MPLS network. Enterprises often use VPLS to provide high-speed any-to-any forwarding at Layer 2 without the need to rely on Spanning Tree to keep the physical topology loop free. VPLS leverages the concept of linking virtual Ethernet bridges using MPLS pseudowires to interconnect sites in a full-mesh topology and form a single logical bridge domain. Compared to traditional LAN switching technology, VPLS is more flexible in its geographic scaling, as the sites may be within the same metropolitan domain or may be geographically dispersed over a region or a nation.
- **Overlay Transport Virtualization (OTV)** —OTV introduces the concept of MAC routing, whereby a control plane protocol is used to exchange MAC reachability information between network devices providing LAN extension functionality. This is a significant shift from Layer 2 switching, which has traditionally leveraged data plane learning, and it is justified by the need to limit flooding of Layer 2 traffic across the transport infrastructure. If the destination MAC address information is unknown, then traffic is dropped (not flooded), preventing wasting precious bandwidth across the WAN.

In addition, OTV introduces the capability of dynamic encapsulation for Layer 2 flows to be sent to remote locations. Each Ethernet frame is individually

encapsulated into an IP packet and delivered across the transport network. This eliminates the need to establish virtual circuits, called pseudowires, between the data center locations.

Finally, OTV enables multihoming with automatic detection. This is critical in increasing high availability of the overall solution, allowing two or more devices to be leveraged in each data center to provide LAN extension functionality without running the risk of creating an end-to-end loop that jeopardizes the overall stability of the design.

- **Radio Aware Routing**—Radio Aware Routing provides a session-based mechanism for sharing radio network status such as link quality metrics and establishing flow control between a router and an RFC 4938–capable radio. RFC 4938 is an IETF standard that defines PPP-over-Ethernet (PPPoE) extensions for Ethernet-based communications between a router and a mobile radio or satellite modem that operates in a variable-bandwidth environment with limited buffering capabilities. The feature enables optimal route selection based on cross-layer feedback and faster convergence when nodes join and leave the network.
- **Redundancy Group Infrastructure**—Redundancy Group Infrastructure provides router-to-router high availability, allowing the configuration of a pair of routers to act as backup for each other. When a failover occurs, the standby router seamlessly takes over. The CSR 1000V supports the Interchassis Asymmetric Routing Support for Zone-Based Firewall and NAT feature, which allows the forwarding of packets from a standby redundancy group to the active redundancy group for packet handling. If this feature is not enabled, the return TCP packets forwarded to the router that did not receive the initial TCP synchronization (SYN) message are dropped because they do not belong to any known existing session.
- **Virtual Extensible LAN (VXLAN)** —VXLAN is a network virtualization technology that solves the scalability problem in large data center or cloud computing environments. VXLAN encapsulates a MAC frame inside a UDP packet. It uses a 24-bit virtual identifier called the VXLAN network identifier (VNID) to provide Layer 2 separation. This 24-bit VNID offers scalability up to 16 million Layer-2 VXLAN segments. The UDP encapsulation enables these segments to be routed across Layer 3 networks using equal-cost multipath routing. The CSR 1000V supports this feature and acts as a Layer 2 and Layer 3 VXLAN gateway. The Layer 2 gateway leverages UDP encapsulation on data plane MAC address learning and forwarding on multi-destination Layer 2 traffic. The CSR 1000V offers interoperability, allowing connectivity between hosts on the VXLAN and hosts on a traditional VLAN network.
- **Zone Based Firewall (ZBFW)** —Zone Based Policy Firewall, also known as Zone Based Firewall, is a stateful inspection firewall running on the CSR 1000V that offers a flexible advanced security model. With ZBFW, router interfaces are assigned to security zones, and the firewall inspection policy is applied to traffic moving between the security zones. ZBFW supports many types of application inspection, including HTTP, Secure HTTP (HTTPS), Secure Shell Protocol (SSH),

Simple Mail Transfer Protocol (SMTP), instant-messaging applications, and point-to-point file sharing. Inter-zone security policy offers the flexibility and granularity for allowing different inspection policies to be applied to multiple host groups connected to the same router interface.

Existing IT staff will be able to configure these features, and using them allows you to extend existing enterprise security into the cloud. You can apply security policies between virtual networks or applications in the cloud as well as between cloud and external locations.

For a complete list of features supported by the CSR 1000V, please refer to the Cisco Feature Navigator at <http://tools.cisco.com/ITDIT/CFN/jsp/index.jsp>.

Summary

Now that you have read this chapter, you should have a fundamental understanding of the system architecture of IOS and the evolution of IOS XE. The chapter provides detailed discussion of the operations and the system architectures for IOS and IOS XE network operating system. The components of IOS XE and the ASR 1000 are fundamental to understand the operation of the CSR 1000V. This chapter has provided the background information you need to go deeper into the CSR 1000V's system architecture and operation.

This page intentionally left blank

Hypervisor Considerations for the CSR

A hypervisor is similar to an operating system in some ways, and it is different in others. The concept of a hypervisor is an important foundation as it relates to the CSR operation. The objective of this chapter is to disambiguate the two so that you have a clear understanding of an OS and a hypervisor. This chapter covers operating system concepts with a focus on Linux because a Cloud Service Router's (CSR) operating system is based on Linux.

Understanding Operating Systems

Computer hardware typically consists of memory, CPU, and I/O devices. They are complex systems, and all computer applications need parts of this hardware. It is possible to dedicate an entire piece of computer hardware to a single application. However, doing so may result in a computing resource idling most of the time. If you choose to run multiple applications, to better utilize computing resources, you may run into a resource contention issue. You need to find a mechanism to allocate the hardware resources to your applications when they need them. To avoid resource oversubscription, there has to be a governor of sorts that acts as the master for scheduling the hardware resources. Applications talk to this governor and get CPU cycles, memory, and I/O resources when they need them. This governor is the operating system.

An operating system is software that manages hardware resources and makes them available to applications. Application developers can write software that talks to the hardware. However, software that interacts directly with the hardware is complex and difficult to code. It is best left to people who understand the inner working of the hardware to write that piece of code. This way, application developers can do what they do best: write applications without having to worry about how their code will be scheduled on the hardware it eventually runs on. That scheduling is the domain of an operating system, which manages and schedules hardware resources for the applications running on it. But how does the operating system do it? Let us start with how an operating system is designed.

Operating System Design

An operating system's design must cover two broad elements: physical resource management and software access to physical resources. The following sections describe these elements and their components.

Physical Resource Management

Most modern hardware is designed to support multiple applications. Multitasking is not possible without the operating system allocating physical resources to multiple applications at the same time. Memory, CPU, and network interfaces are some very common hardware resources that need to be made available to applications. The OS schedules the CPU cycles so that each application gets a piece of the CPU when it requires it. The OS also needs to manage memory from the physical memory available. Random access memory (RAM) is the memory that operating systems or applications use for faster access by the processor.

An OS can schedule the CPU in different ways. At a given instant, most CPUs can service just one process. With multiple applications competing to get a piece of the CPU cycles, the OS must service the requests in such a way that each application gets a slice of the CPU cycle. An OS can achieve this in several ways:

- **FIFO (first in, first out)**—In this simplest of scheduling algorithms, a process is serviced by the CPU on a first-come, first-served basis. It is fairly easy to implement this in the C language. You just create a circular list to have the OS remove the process from the front of the queue, run it until completion, and allow the next process to take the CPU. The advantages of this kind of implementation are that it is easy to implement and simple to understand. On the downside, however, short processes at the tail of the queue have to wait a long time to get serviced as the non-preemptive implementation waits for each process to complete before moving on to the next.
- **Round-robin scheduling**—This is a legacy scheduling mechanism wherein each process is given a fixed amount of CPU time. If the process does not get completed within the allocated time, the process is moved to the end of the queue, and the next process is serviced. If the process gets completed, there is a way to relinquish the time it no longer needs from the allocated quota. This is a good option for some real-time environments. It gives very good performance, especially for embedded applications with simple schedulers, and enforces strict fairness, so no process ever gets starved. The round-robin method, however, suffers from multiple drawbacks. For example, the system becomes too slow when the CPU time slice allocation is too small and a lot of processes are waiting to be serviced. If you increase the CPU time slice allocated for each process, the system becomes unresponsive. The advantage of this kind of implementation is the simplicity: It's easy to comprehend and code.
- **SPN (Shortest Process Next)**—In the SPN implementation, the CPU schedules the shortest process first, so the process that takes the least time to get executed is scheduled first. In this algorithm, the OS needs to know the exact time each process takes to get done, and the user provides this input. So if the input isn't accurate, the

efficiency of the system suffers. With multiple short jobs, the long ones get queued up for a while.

- **SRT (Shortest Remaining Time)**—SRT is a better version of the SPN algorithm. Here the process that can be completed in the shortest time jumps to the top of the queue. So whenever there is a process that requires the least amount of CPU time, it cuts the queue of scheduled processes and goes right to the start of the queue. This implementation suffers from the same drawbacks as the SPN algorithm.
- **Preemption**—This algorithm allows an OS to relinquish the CPU (currently processing a task) in favor of a higher-priority process. This means the OS has the ability to preempt. In case of handling interrupts, scheduling is stopped until the interrupt is taken care of. For example, the scheduler in the Linux kernel is invoked regularly within a stipulated time (such as after each timer interrupt). When called, the scheduler picks up the next process it must service, based on priority and other factors. This model yields better OS scalability and response.
- **Priority scheduling**—In this algorithm, each process is assigned a priority, and the higher-priority processes are executed first. Some implementations use the time the process has waited to increase the priority of the process; this prevents the non-priority processes from being blocked.

As with CPU, memory is another resource that needs to be shared between applications. The given physical memory (RAM) is managed by the OS. With limited RAM address space, operating systems will either use segmentation or paging to virtually increase this address space:

- **Segmented memory**—If an operating system opts for segmented memory allocation, it uses a memory management unit (MMU), which is a hardware device that translates a logical address (segment with an offset) to a physical address on the memory chip. This allows computers to present to applications more memory that is addressable than what is physically available on the memory chip.
- **Paged memory**—Segmented memory is not very popular with modern operating systems, which tend to use paged memory. With paged memory, the MMU is used to translate virtual addresses to physical addresses. When using this model, an OS can map multiple 4KB chunks or pages in most x86 architectures or 4MB pages with a huge pages option turned on. So you can have data at an offset of 0x200 in the physical memory mapped to an address XYZ (at an offset of 4GB) in virtual memory even though your RAM isn't 4GB. This essentially does two things. First, it gives your application a feeling of operating with much more memory than is available physically. Second, it allows the OS to give each process its own address space (albeit virtual address space) that makes the process self-contained within that address space. Everything else is hidden from the process, and it does not have access to addresses outside the virtual address space allocated to it. This prevents it from corrupting the data of other processes and enhances security. Figure 3-1 details the memory paging concept. It illustrates how a virtual address is mapped to a physical location on the RAM.

In most Linux versions, the kernel uses the preemption model when it comes to CPU scheduling. For memory management, the kernel uses paging.

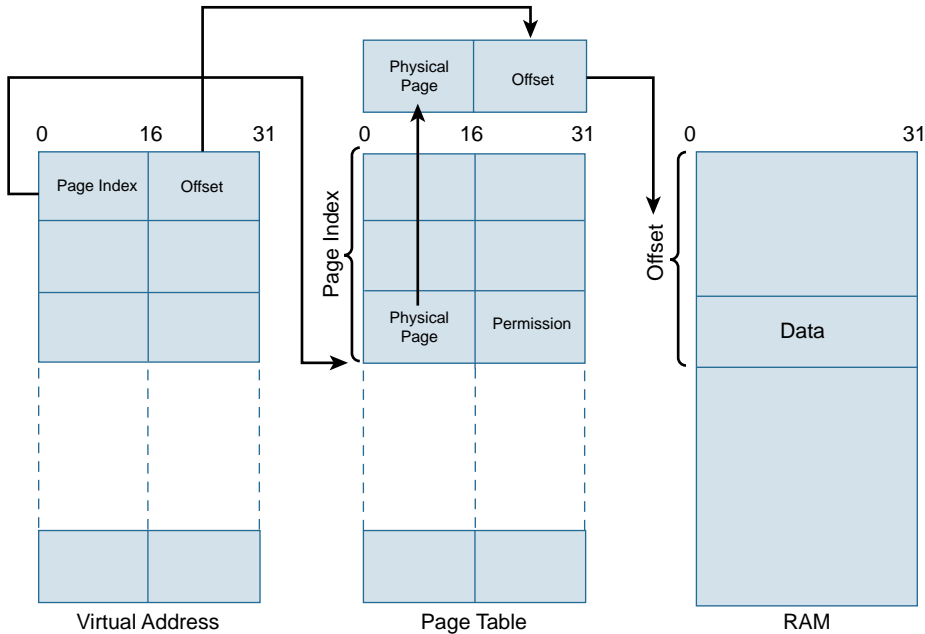


Figure 3-1 *Memory Paging Concept*

Software Access to Physical Resources

Every OS needs to provide a software interface for the applications to access the hardware. We discussed the design options available for operating systems when managing hardware. An OS must provide interfaces for applications to access the hardware resources it is managing. This is typically done through an application programming interface (API). It allows an application developer to access the hardware through software handles. The OS should also give the user an interface to access the OS software through a shell or a graphic interface. This allows the user to use and monitor the OS.

Now that we have discussed the design options available when implementing an OS, let us discuss the different components that are building blocks for an OS:

- The kernel is at the core of the operating system.
- Libraries provide services to the applications that run on the operating system.
- Drivers allow the software to use the hardware.
- A boot mechanism loads the operating system into the main memory.

Kernels

A kernel is at the heart of an operating system. It is the piece of code that sits closest to the hardware. The kernel is an integral part of an operating system, and its duty is to bring all devices to a known state and make the computer system ready to be used.

The kernel code provides a layer of abstraction between the hardware and the software running on the system. Through APIs, applications have the kernel perform hardware jobs. Kernels allow the hardware to be shared between multiple applications, and kernels overcommit resources to applications. (You'll learn more about the Linux implementation later in this chapter.)

When UNIX was originally designed, it had a simple design wherein the entire operating system code was one big layer, as shown in Figure 3-2. This was a good approach at the time, given the limited number of programmers, and it kept things simple.

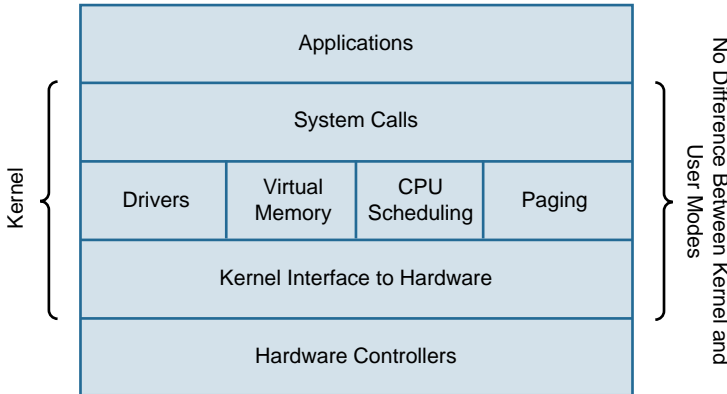


Figure 3-2 *Original UNIX System Structure*

Microkernels

As the demand on the OS increased, there had to be a more modular way to structure the design of the kernel. Microkernels were introduced to strip off all nonessential components of the kernel code and move those components to system applications; this kept the kernel code small, sleek, and efficient. Thus the kernel space and the user space were segregated from one another; the user space became the code that runs outside the kernel. The drivers, file systems, and other services are all part of the user space. The microkernel code provides hardware management (memory and CPU) and messaging between services. This way, the kernel code is kept intact, and the subsequent enhancements do not involve rebuilding the kernel.

The object-oriented approach to programming brought changes in operating system design, too. Take, for example, Solaris. It has a small kernel and a set of modules that can be dynamically linked to the kernel. The kernel can therefore be small and does not need to take care of interprocess communication (IPC) because the modules are free to contact each other.

Hybrid Kernels

Most of today's operating systems are a hybrid of the legacy architecture and modern object-oriented approach. Take, for example, Android, shown in Figure 3-3.

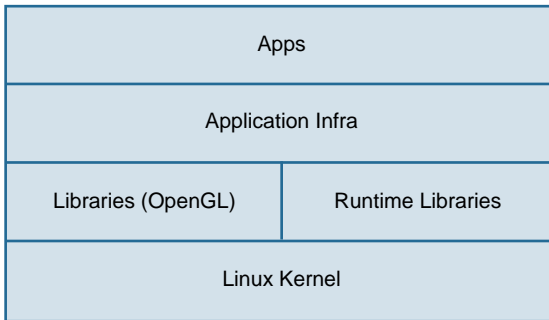


Figure 3-3 *Android Architecture Approach*

In the Android approach, the Linux kernel is used as is. The Linux kernel runs libraries like OpenGL and WebKit, and it links to certain Android runtime libraries dynamically at runtime. The runtime libraries allow multiple instances of containers to be created simultaneously, providing security, isolation, memory management, and threading support. This runtime support provides improved compilation and runtime efficiency to applications compared to traditional operating systems.

The Cisco IOS Kernel

The IOS kernel manages system resources such as scheduling and memory management. However, it's different from other modern kernels in that it doesn't have segregation between the kernel and user mode. Everything in IOS, including the kernel, runs in user mode, as shown in Figure 3-4. This means that all processes have full access to system resources.

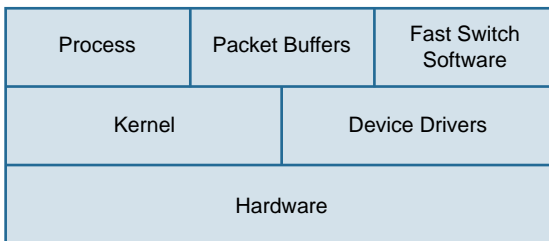


Figure 3-4 *IOS Architecture*

The kernel in IOS mainly runs the scheduler and the memory manager.

The Scheduler

The IOS scheduler has three main categories of process queues that hold context information. The scheduler moves the process context from one queue to another, based on the state of the process. Following are the queues:

- **Idle queue**—Holds processes that are waiting on an event to run.
- **Dead queue**—Holds processes that have been killed but still need to free up resources.
- **Ready queue**—Holds processes that are run ready. There are four ready queues, based on priority:
 - **Critical**—The scheduler first empties this queue. The processes in the critical queue are run until the queue is empty. Only then does the scheduler schedule processes from the other queues.
 - **High**—After all the critical processes have completed, the scheduler picks up processes in the high queue. The scheduler checks the critical queue in between running processes from the high-priority queue.
 - **Medium**—After all high processes have been executed, the scheduler picks up the processes in the medium queue. Here, too, in between running medium-priority processes, the scheduler looks for processes in the high-priority queue (and this is repeated if there are processes in the critical queue).
 - **Low**—The same algorithm is followed for the low-priority queue as for the other queues. In between running low-priority processes, the scheduler looks for processes that are ready in the medium, high, and critical queues.

`show process CPU` is the command-line command for checking what processes are running on the CPU.

The Memory Manager

The IOS memory manager is responsible for managing the memory available to IOS. The memory manager within IOS has three components:

- **Region manager**—This allows parts of IOS to create regions in the memory. It allows querying memory regions to determine the amount of memory available.
- **Pool manager**—Each `malloc` or `free` from a process has to go through the pool manager. It maintains the free memory information from each pool. The pool manager coagulates discontinuous freed-up areas within a memory pool. It tries to allocate a continuous area of memory for each process that asks for memory. Whenever a process frees up memory fragments, the pool manager tries to coagulate the freed memory into continuous memory areas to be made available to processes. Every memory address managed by the pool manager has 32 bytes of overhead associated with it, because the pool manager gets its memory from multiple blocks of free memory of varying sizes.

- **Chunk manager**—The IOS chunk manager manages a large number of randomized blocks of memory. It uses a single 32-byte overhead for a large number of small freed-up memory segments. When a process requests a large block of memory, the chunk manager fragments this memory chunk into smaller fixed-sized blocks, which are then used by the process. The advantage here is that there is just one 32-byte overhead associated with a large memory block (and the chunk manager later fragments it).

IOS scheduling and memory management processes are part of the IOS kernel. However, all kernel processes and non-kernel processes run in the user space in IOS. IOS was designed to run on a fixed set of hardware (Cisco routers), and hence this design is easy to implement and effective, too.

You will see in Chapter 4, “CSR 1000V Software Architecture,” how the IOS XE design uses a Linux kernel to do all the kernel activities, while IOS runs as a process on a Linux kernel. The design of IOS XE facilitates portability of code between hardware, and the implementation is similar to that of a hybrid kernel.

The Boot Process

To boot any computer system, you must first bring to life basic input/output devices. Then you need to locate and execute code to load drivers to power up all devices connected to the computer. Subsequently, you load the kernel and the operating system. Here we take Linux as an example to look at how the boot process works. It is important to understand the boot process when trying to get familiar with Cisco’s CSR because it is based on IOS XE, which uses a Linux kernel.

On cold start, the computer system first has the boot loader program loaded into a known place in memory. The boot loader program is located in the beginning of a hard drive or partition. (The following section details the steps involved in booting.) The boot loader performs tasks such as initializing the CPU, identifying key pieces of hardware, and executing the kernel code. Boot loader code may or may not be part of the operating system. For example, in Windows NT, the boot loader is a part of the operating system. In the case of the Cisco CSR, GNU GRand Unified Bootloader (GRUB) is used to accomplish boot loader functions and isn’t a part of the operating system.

It is important to understand how a computer system boots up its different components. The operating system becomes the master of the computer resources and provides hardware resources to the applications running on it. But how does the operating system get on to the computer? A clear understanding of this will help you debug and troubleshoot CSR issues.

There are six essential steps involved in loading an operating system to a computer and making it ready to run the applications. Following are the steps in the Linux boot process; the CSR boot process follows this sequence as well:

- Step 1.** The Basic Input/Output System (BIOS) runs from the ROM and is OS independent. It performs a power-on self-test (POST), which is a basic check for fundamental hardware on the computer system. BIOS looks for the Master Boot Record (MBR), loads it, and executes it.
- Step 2.** The MBR is located in the first sector of the bootable disk. It is `/dev/sda` on the Linux kernel that is used with the CSR. On most Linux kernel implementations, it is smaller than 512 bytes in size. The first 446 bytes provide the primary boot loader info. The next 64 bytes contain the partition table info, and the last 2 bytes are an MBR validation check. The main job of the MBR is to load and execute the GRUB boot loader.
- Step 3.** GRUB gives you an option of multiple kernel images to be executed on a splash screen. You can choose the kernel images or let it load the default in a few seconds.

On the CSR, GRUB gets its menu of kernel images from `menu.lst`, located under `/boot/grub/`, as shown in Example 3-1.

Example 3-1 *Kernel Images Available to GRUB Are Listed in menu.lst*

```
cat /boot/grub/menu.lst
default 0
timeout 5
serial --unit=0 --speed=9600 --word=8 --parity=no --stop=1
terminal --timeout=10 serial console

fallback 1

title CSR1000v-packages.conf
    root (hd0,0)
    kernel /packages.conf rw root=/dev/ram console=ttyS1,9600 max_loop=64
    HARDWARE=virtual SR_BOOT=bootflash:packages.conf

title CSR1000v-GOLDEN IMAGE
    root (hd0,4)
    kernel /csr1000v- csr1000v-universalk9.03.10.00.S.153-3.S-ext.SPA.bin
    rw root=/dev/ram console=ttyS1,9600 max_loop=64 HARDWARE=virtual
    SR_BOOT=bootflash:csr1000v-universalk9.03.10.00.S.153-3.S-ext.SPA.bin
```

So the main job of GRUB is to load and execute the Kernel and `initrd` images.

- Step 4.** The kernel mounts the root file system and executes the `/sbin/init` program. Because `init` is the first program to be executed by the Linux kernel, it gets a process ID of 1:

```
ps -ef | grep init
root      1      0  0  2014 ?        00:00:18 init [3]
```

The main jobs of the kernel are to initialize the devices attached to the computer system, mount the root file system, and run the `init` process. To load the drivers, however, the kernel must have a file system. Before the root file system is initialized, the kernel needs a file system to make sure the kernel modules are loaded appropriately. This is done by the `initrd` (or initial ramdisk) file system. This is a temporary root file system that the kernel uses when it boots up. After the kernel boots and loads the main root file system, it takes `initrd` offline.

Step 5. The kernel runs the `init` process, which creates all the processes from the script located in the file `/etc/inittab`. The main job of the `init` process is to set up the user space. It essentially decides the Linux run level. After it is done creating all the processes from `/etc/inittab`, it goes into a wait state until one of three events occurs:

- Processes die
- Power failure
- Request via `/sbin/telint` for a change in run level

These are the run levels used by most Linux implementations:

- 0—Halt (Do not set `initdefault` to this.)
- 1—Single-user mode
- 2—Multiuser mode, without NFS (the same as 3, if you do not have networking)
- 3—Full-multiuser mode
- 4—Unused
- 5—X11
- 6—Reboot (Do not set `initdefault` to this.)

Step 6. The run level programs/services get started while your Linux system boots. Depending on your `init` level setting, the system executes one of the following for your run level:

- 0—Halt: The program executed is `/etc/rc.d/rc0.d/`.
- 1—Single-user mode: The program executed is `/etc/rc.d/rc1.d/`.
- 2—Multiuser mode: The program executed is `/etc/rc.d/rc2.d/`.
- 3—Full-user mode: The program executed is `/etc/rc.d/rc3.d/`.
- 4—Unused mode: The program executed is `/etc/rc.d/rc4.d/`.
- 5—X11 (full multiuser with GUI): The program executed is `/etc/rc.d/rc5.d/`.
- 6—REBOOT: The program executed is `/etc/rc.d/rc6.d/`.

Under the `/etc/rc.d` directories you find programs that start with *S* and *K*. *S* stands for *startup*, meaning that these are executed during startup. *K* stands for *kill*, meaning these programs are executed during a system shutdown.

On a CSR, the programs shown in Example 3-2 are executed on run level 3 (full-multiuser mode).

Example 3-2 *Programs That Are Executed on the Full-Multiuser Run Level*

```
[CSR:/etc/rc.d/rc3.d]$ ls -lrt
total 212
-r-xr-xr-x 1 root root 4684 Dec 2 15:45 S60nfs
-r-xr-xr-x 1 root root 4131 Dec 2 15:45 S56xinetd
-r-xr-xr-x 1 root root 4958 Dec 2 15:45 S55sshd
-r-xr-xr-x 1 root root 2778 Dec 2 15:45 S28autofs
-r-xr-xr-x 1 root root 2164 Dec 2 15:45 S26pcscd
-r-xr-xr-x 1 root root 1870 Dec 2 15:45 S20virt_support
-r-xr-xr-x 1 root root 8092 Dec 2 15:45 S14netfs
-r-xr-xr-x 1 root root 2615 Dec 2 15:45 S13portmap
-r-xr-xr-x 1 root root 1369 Dec 2 15:45 S12syslog
-r-xr-xr-x 1 root root 10115 Dec 2 15:45 S10network
-r-xr-xr-x 1 root root 7460 Dec 2 15:45 S08iptables
-r-xr-xr-x 1 root root 220 Dec 2 15:45 S99local
-r-xr-xr-x 1 root root 135553 Dec 2 15:45 S80binos
```

Linux Memory Management

The Linux kernel performs memory management. The following sections cover the key concepts involved in Linux kernel memory management.

Linux Swap Space and Memory Overcommit

As discussed earlier, virtual memory is the memory that is allocated to a process when it comes in asking for a `malloc()`. (`malloc()` is a subroutine in the C programming language for performing dynamic memory allocation.) This virtual memory is mapped to a physical memory location. Whenever a processor executes a program instruction set, it reads the instructions from a virtual memory location in a virtualized environment. However, before this can be done, the processor has to map the virtual memory location to a physical address. This mapping is done using page tables.

This method of memory allocation and addressing is inherently subject to overcommitment, as shown in Figure 3-5. It means that the operating system can allocate more virtual memory than is physically available on the RAM. This is based on the assumption that the process/virtual machine asking for memory will not need all the memory requested to start with. In a virtualized environment, this means you can create more virtual machines than your RAM prescribes. This is a very powerful technique, but

it needs additional memory management to make sure processes/virtual machines get physical memory when they actually need it.

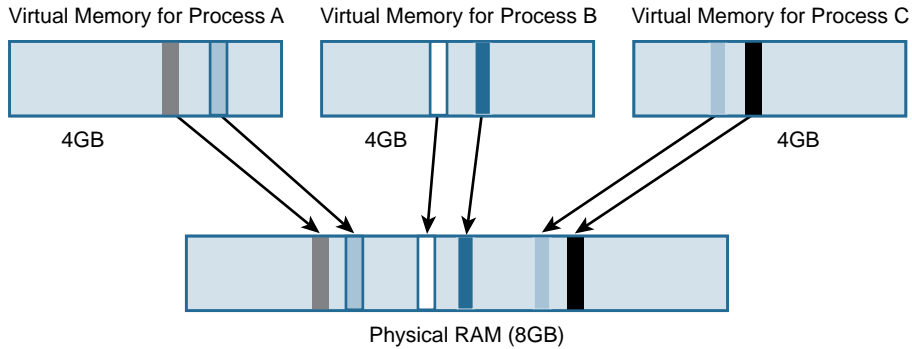


Figure 3-5 *Memory Overcommitment*

Figure 3-5 shows 8GB of physical memory available and the memory used by the virtual process mapped to the physical memory. You can, in theory, create two 4GB virtual machines on this. However, because two virtual machines will not use the entire 4GB physical address space, it's better to overcommit and use the memory for hosting another VM. If there is a surge in the memory requirement, swapping is used to manage the spike, as shown in Figure 3-6.

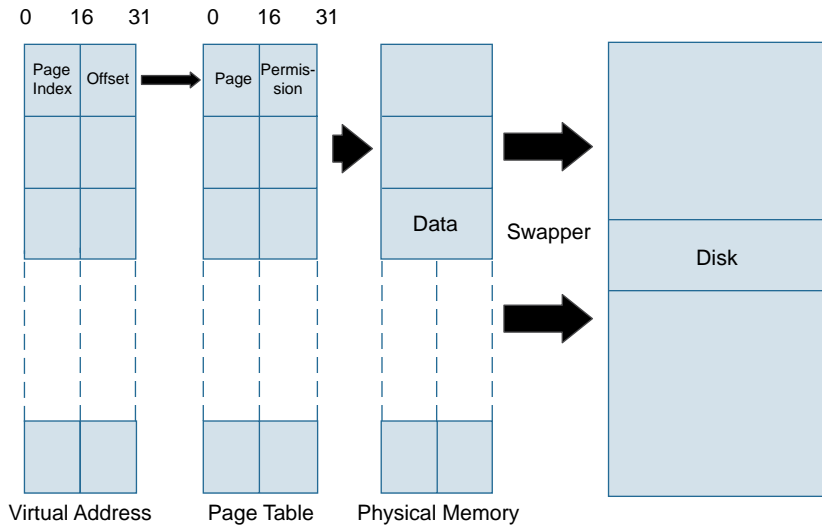


Figure 3-6 *Memory Swapping*

With overcommitting of memory, you run the risk of having a process or virtual machine want physical memory when the physical RAM space is all used up. In such a case, Linux has to discard a page residing in RAM to accommodate the new request. If the page to be discarded is not written into, it can just be removed as it can be easily brought back when the processes need it. If a page in the cache that has been written into (a dirty page) needs to be removed (which is done when there are no empty pages left), Linux removes a dirty page from RAM and puts it in a special file called a *swap file*. This is usually on a secondary storage device of a computer system like a hard disk. This process is called *swapping*.

Now the obvious question is, how does Linux decide which dirty page to swap? Linux uses the least recently used algorithm to decide which dirty page can be swapped. Each page has an age associated with it. The pages that are being accessed are “young” pages, while the pages that are not accessed a lot get “old.” When Linux has to choose which dirty page to swap, it chooses the oldest page.

A swap algorithm must be effective; otherwise, you can run in to a situation where your CPU becomes too busy swapping and is unable to give time to processing real workload. Such a situation is called *thrashing*.

Linux Caching

Disk access is time-consuming, and to speed up the access, Linux deploys a cache mechanism by which it reads from the disk once and then stores the entry in memory (the Linux swap cache) for subsequent access. A *swap cache* is a list of page table entries of the swapped-out pages’ locations in the swap file. Consider a situation where a dirty page is swapped. It is then brought back to memory unmodified. Now if there is a need to swap this page again, Linux does not push it back to the swap file. Instead, it simply discards the page because the swap file already has a copy of the page.

As discussed earlier, an OS maps the virtual address to a physical address using page tables. These table entries are stored in a hardware cache. This hardware cache has the translational look-aside buffer (TLB). When a processor wants to map a virtual address to a physical address, it gets the page table entry from TLB. If the processor finds the information it is looking for, it gets the physical address from it. However, if it cannot find the entry there, it asks the OS to update the TLB, using an exception. The OS subsequently updates the TLB and clears the exception.

Understanding Hypervisors

As discussed in Chapter 1, “Introduction to Cloud,” virtualization allows you to fully utilize modern hardware that would otherwise not be fully utilized. Just as an operating system makes hardware resources available to the applications running on it, a manager is needed to allocate the hardware resources to virtual machines running on it. This is the job of a virtual machine manager (VMM), which is a software layer that sits between the hardware resource and the virtual machines. VMM makes sure the virtual machines get the hardware resources they require.

As mentioned in Chapter 1, virtualization is not new concept. In 1965, IBM engineers developed software that allowed the IBM360/65 to share memory with the 7080 emulation that ran on the IBM360. Engineers were essentially trying to access memory that the operating system would normally deny access to. Because operating systems were referred to as *supervisors*, the term *hypervisor* was born for this piece of code that allowed engineers to override the supervisor. The term *hypervisor* then replaced *VMM*, but essentially the term hypervisor can be used interchangeably with *VMM*.

How Does a Hypervisor Compare to an Operating System?

Now that you know how the operating system shares hardware resources, we can compare it to a hypervisor. Broadly speaking, a hypervisor is a piece of software that provides operating system services to virtual machines running on it. A type 1 hypervisor runs over bare-metal x86 hardware architecture, as an operating system does, but it also enables other operating systems to run on it. An operating system creates software handles that an application uses to access the hardware resources. With a hypervisor, these software handles enable the user to run not just applications on it but also self-contained operating systems that think they run on bare metal (unless, of course, in a para-virtualized system).

Note Computer operating systems provide multiple levels of access to shared resources. There is therefore a hierarchy of privileged resource access in the system. In x86, the term *ring* is used to denote the hierarchy of access. Note that ring 0 is the highest privilege level, and the lowest privilege level has the highest ring number. These are some important features of x86 ring architecture:

- Ring 0 has the most privileges and interacts directly with the physical hardware, such as the CPU and memory.
- Ring 1 and 2 aren't used by most chipset architectures, and most chipsets support just two modes (such as ring 0 and ring 3).
- Ring 3 runs the user mode and has the lowest privilege level.

When a process running in the user space (ring 3) wants to make a privileged call to I/O devices (shared resources), it makes a system call to the kernel. The kernel runs a snippet in ring 0 after receiving the request from the user space to grant this access based on the security restriction of the drivers. (This is achieved using the `SYSENTER/SYSEXIT` instruction set, available on Pentium II+ processors.)

Now consider the case where there are multiple operating systems running on an x86 platform. All processes in the operating system make system calls as if the hardware belongs to them. So there needs to be a way to make sure that an application running on a guest OS does not trample over system calls from another OS. This is where hypervisors come in. Because a hypervisor is aware of all the operating systems running on it, it ensures that the kernel calls are prioritized and one system call does not conflict with another, as shown in Figure 3-7. Just as a process in user mode makes system calls to the kernel, a guest OS calls the hypervisor to execute privileged instructions on the chipset. These processes are called *hypercalls*.

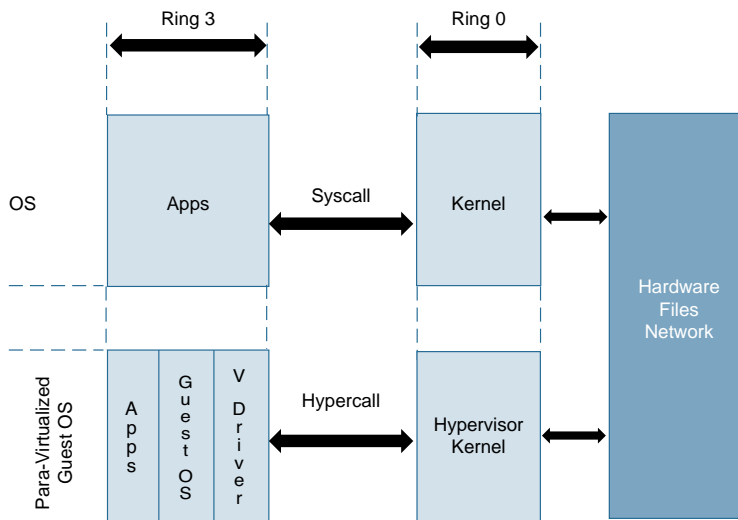


Figure 3-7 *Syscall Hypercall*

The previous discussion comparing operating systems and hypervisors assumes that the operating systems running as guest operating systems are completely oblivious to the fact that they are not the only ones running on the x86 platform. However, you can make an OS aware that it is running on a hypervisor and not bare metal (via para-virtualization, introduced in Chapter 1). The enlightened OS kernel uses hypercalls to communicate with the hypervisor. With the knowledge that it is running on a hypervisor, the enlightened OS kernel has optimized code, and this gives it better performance than unenlightened guest operating systems.

Note Intel x86 platforms starting with the Pentium II series chipset have a `SYSENTER/SYSEXIT` instruction set that enables faster access to user-land processes to access the kernel.

Intel's 64 and IA-32 architecture software developer's manual describes `SYSENTER` (a fast system call) as follows:

Executes a fast call to a level 0 system procedure or routine. `SYSENTER` is a companion instruction to `SYSEXIT`. The instruction is optimized to provide the maximum performance for system calls from user code running at privilege level 3 to operating system or executive procedures running at privilege level 0.

The same software developer's manual describes `SYSEXIT` (the fast return form of a fast system call) as follows:

Executes a fast return to privilege level 3 user code. `SYSEXIT` is a companion instruction to the `SYSENTER` instruction. The instruction is optimized to provide the maximum performance for returns from system procedures executing at protection level 0 to user procedures executing at protection level 3. It must be executed from code executing at privilege level 0.

Type 1 Hypervisor Design

As discussed in Chapter 1, type 1 hypervisors run on bare metal, without any operating system underneath. Type 1 hypervisors have direct access to hardware and hence provide better performance than type 2 hypervisors (which run on an OS).

Type 1 hypervisor architectures can be classified broadly into two categories, monolithic architecture and microkernel architecture, as described in the following sections.

Monolithic Architecture

As is the case with operating system design, monolithic architecture is present in hypervisor design, too. The hypervisor code using this design includes just one instance of a virtualization stack and supports multiple instances of guest operating systems run on it. Because all the device drivers are in the kernel, which is the supervisor area, VMs are given a common pool of virtualized drivers to choose from. There can be no guest-specific drivers in this design.

Microkernel Architecture

A microkernel approach strips down the actual hypervisor software to essential calls (like kernel code) and pushes the other components to a management guest operating system running on the kernel. The kernel talks to the hardware, and the management guest OS runs the drivers and virtualization stack. The management guest OS handles I/O calls for all other guest operating systems. This gives console access to the VM.

Xen and Hyper-V are examples of microkernel design. Figure 3-8 compares monolithic and microkernel architectures.

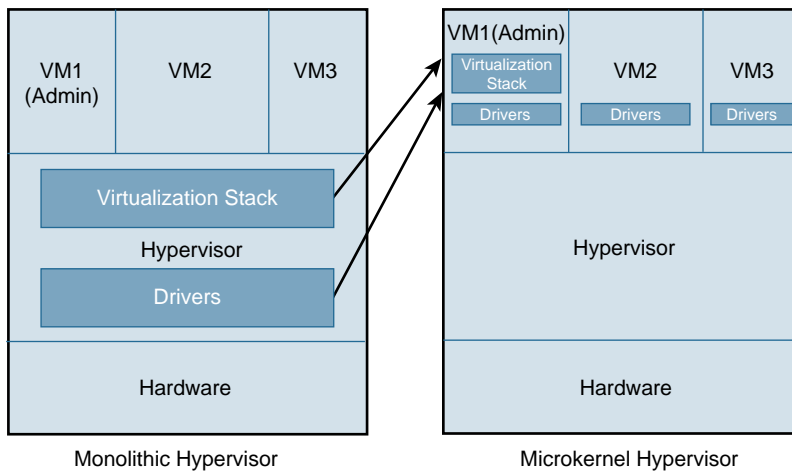


Figure 3-8 Monolithic/Microkernel Comparison

Core Partitioning

Core management is one of the key aspects to consider when selecting a hypervisor. The basic approach is static partitioning, wherein each VM is permanently dedicated to a core. In dynamic partitioning, the VMs are allowed to migrate between cores. This is critical when there are more VMs than there are cores. The hypervisor must optimally schedule the cores for the VMs. This means timesharing the cores between multiple VMs. Take, for example, a two-core system. Without core partitioning, a hypervisor can allocate two cores to a single VM or single core per VM (assuming that there is no hyperthreading, in which a single processor is split into two logical processors and each processor then has the capability of executing instruction sets independently). With core partitioning, the hypervisor can allocate two cores to two VMs because the cores are time shared and not dedicated to a single VM.

The following sections cover ESXi and KVM, the two major type 1 hypervisors in the industry today, and provide a brief overview of Hyper-V and Xen.

ESXi Hypervisor

ESXi from VMware is one of the popular hypervisors in the enterprise data center environment. As a type 1 hypervisor, it runs on bare metal and provides the user a platform to create a virtual infrastructure.

Architectural Components of ESXi

ESXi architecture mainly comprises the following components:

- The VMkernel
- Processes running on the VMkernel
- Device drivers
- File systems
- Management

The VMkernel

In the early days, ESXi was called ESX, and at that point it used a Linux kernel. But after version 4.1 of ESX, VMware stopped the development of ESX in favor of ESXi, which did not have a Linux kernel. On an ESX hypervisor, the Linux kernel boots first, and then a script loads the VMkernel. In versions since ESX 3, the Linux kernel loads the VMkernel from `initrd`. With ESXi, VMware got rid of the Linux-based service console. This resulted in a smaller memory footprint (reduced by 90MB) and allowed ESXi to be embedded within the host flash, thereby eliminating the need for a local boot disk.

The VMkernel is a POSIX-like OS that performs OS functions. It is, however, designed with virtualization in mind. The design supports running multiple virtual machines by providing functionalities like resource scheduling, I/O operations, and handling of device drivers.

Components of the VMkernel

As a type 1 hypervisor, ESXi controls all the hardware resources. The VMkernel is responsible for carrying out the primary operating system tasks, which include CPU scheduling, memory management, and I/O operations:

- **CPU scheduler**—The CPU scheduler’s role with the VMkernel is the same as that of a CPU scheduler in an operating system. In a conventional operating system, the CPU scheduler allocates processes or threads to processors by using fairness, responsiveness, and scalability as major design criteria. In the VMkernel, the CPU scheduler does the same thing, but the processor in the case of a hypervisor is a vCPU (virtual CPU). A vCPU is an execution context or a series of time slots on a processor. In a multicore environment, the scheduler splits these time slots over multiple cores. It has to coordinate time slots between multiple physical cores, and this means overhead. Therefore, just adding more vCPU does not guarantee improved performance.
- **Memory management**—ESXi needs to maintain the translation between the guest operating system’s physical memory and the host’s physical memory. ESXi does this by using physical memory mapping data structures (`pmap`) for each VM. An application running on a guest OS is presented with physical memory from the guest OS. This slab of memory is referred to as a guest’s *physical memory*. The guest OS’s memory page table has a mapping of the guest application’s virtual memory to the guest’s physical memory. However, ESXi lives under the guest OS and gives it virtualized resources. ESXi uses `pmap` and shadow page table to map the guest physical memory address to the host’s physical memory. The shadow page table maintains consistency between the guest’s page table (which maintains the guest virtual memory to guest physical memory mapping) and the `pmap` data structure (which maintains the guest physical memory to the host’s physical memory mapping). The obvious question that comes to mind here is, why is an additional level of indirection needed within the `pmap` data structure? The answer lies in the fact that ESXi manages a virtualized environment, and this additional level of mapping allows it to remap a guest application’s physical memory, on the host, without the guest application knowing about it. Recently, hardware manufacturers, such as Intel and AMD, have eliminated the need for having an additional shadow page table in software by synchronizing the guest page table and `pmap` in hardware.

Figure 3-9 shows the memory addressing mechanism in ESXi.

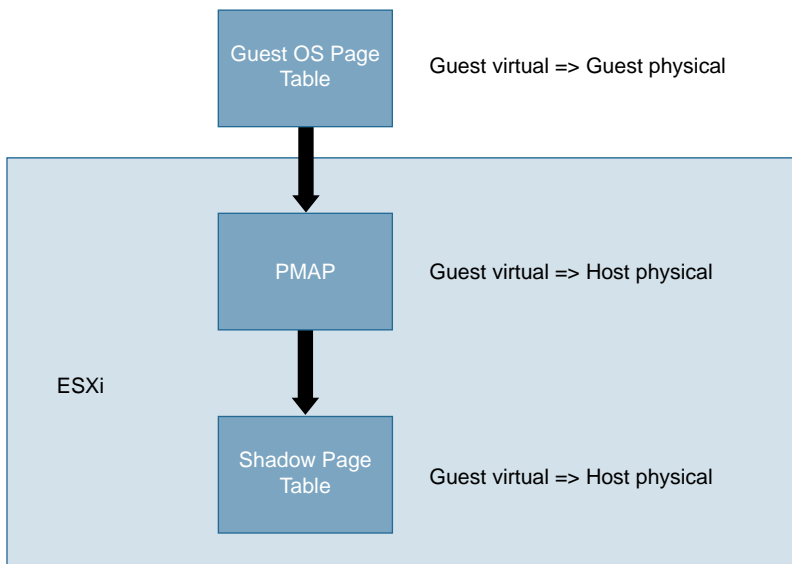


Figure 3-9 *Memory Addressing in ESXi*

Processes Running on the VMkernel

The following processes run on the VMkernel:

- **Direct Console User Interface (DCUI)**—This configuration and management interface, accessible through the server console, is used for initial configuration. This DCUI provides the following services:
 - Sets the root password on the ESXi host
 - Configures the management interface after assigning a NIC to it
 - Starts/stops the management services
 - Views system logs
- **VMM and VMX processes**—Each virtual machine is assigned a VMM (virtual machine monitor) process and a VMX (helper) process. These processes are responsible for providing an environment for virtual machines to run on. These processes provide an environment where the guest OS interacts with the virtual hardware provided to it. VMM passes storage and I/O requests to the VMkernel. Other requests are passed to the VMX process. VMX handles user interface communication, snapshot managers, remote consoles, and I/O requests to devices that are not critical to performance.
- **CIM (Common Information Model) system**—The CIM system allows remote applications to manage hardware using a set of APIs. ESXi depends on this for hardware monitoring and for keeping tabs on system health. CIM is an open

standard that provides a framework for monitoring hardware resources for ESXi. The CIM framework consists of the following:

- **CIM providers**—A CIM provider is a piece of code that enables monitoring and management capabilities of a piece of hardware. A CIM provider is an extremely lightweight plugin that is written for the purpose of monitoring and managing a piece of hardware. Hardware and software makers chose one of the several predefined XML schemas to give management information about their products. CIM providers run inside the ESXi system.
- **CIM manager/broker**—Also called CIM Object Manager (CIMOM), the CIM manager/broker takes in information from all the CIM providers and makes it accessible via a set of standard APIs.
- **hostd**—This is a very critical process. Together with `vpwa` (discussed later in this chapter), it controls management access to the ESXi host. `hostd` is the process responsible for communication between the VMkernel and the outside world. It authenticates users and tracks user privileges. `hostd` talks to the VMkernel and invokes all management operations on VMs, storage, and networking. `hostd` is used by the vSphere API to make a connection to the ESXi host.
- **vpwa**—This process connects to the vCenter. It acts as an intermediate process between `hostd` and the vCenter. (The vCenter and other management aspects of the architecture are covered later in this chapter.)

Device Drivers

In the early days of ESXi, VMware used drivers derived from Linux. This meant that ESXi could support many hardware devices. However, to use these drivers from Linux, ESXi required a mechanism to make these drivers talk to VMkernel. VMkernel is not a Linux kernel, and so VMware had to write an additional layer of software to make these Linux drivers talk to the VMkernel. This additional shim layer, `vmklinux`, provided VMkernel with the APIs needed to communicate to the Linux-based drivers. The downside with this architecture was an additional layer of indirection that degraded the performance. ESXi was also limited by the capabilities of the Linux drivers.

With ESXi5.5, VMware decided to do away with the `vmklinux` shim layer. This means that a driver, native to ESXi now talks to the hardware and has APIs interacting directly with the VMkernel. The new native driver architecture means better performance as it removes the overhead processing of the additional shim layer. You also get better debugging capabilities because VMware can now develop debugging tools for drivers. ESXi5.5 is backward compatible and supports legacy Linux drivers.

Figure 3-10 compares legacy drivers with native drivers.

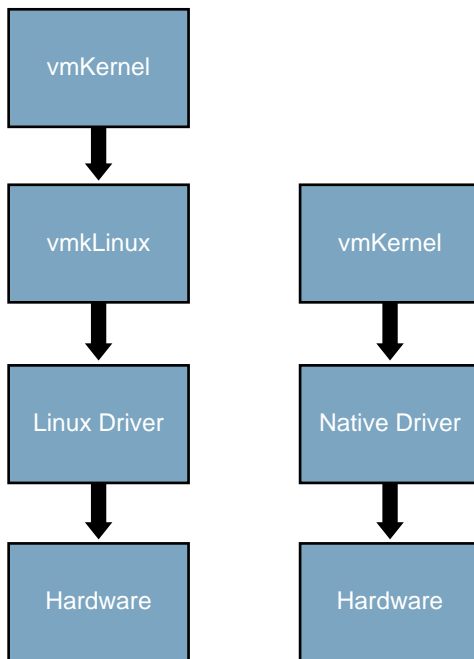


Figure 3-10 *Legacy and Native Driver Architectures*

File Systems

VMware uses a high-performance clustered file system called Virtual Machine File System (VMFS). VMFS is custom built for virtualized environments and enables high-speed storage access for virtual machines. VMFS manages virtual machine storage by creating a subdirectory for each virtual machine and then storing all its contents within that directory. The location of the directory is VMhome.

VMFS is a clustered file system, which means it is mounted simultaneously on multiple ESXi servers. You can therefore access the same set of virtual machines from different servers. VMFS stores each virtual machine file as a virtual machine disk (VMDK) file on a large logical unit number (LUN), which identifies a logically separate storage device that is addressable via SCSI, iSCSI, or Fibre Channel protocols. LUN is typically used with RAID, wherein a group of disks are presented to the host as one logical entity that is a mountable volume. With this logical unit addressable via SCSI, VMFS provides SCSI access to virtual machines. VMFS leverages shared storage to enable multiple ESXi hosts to write to the VM storage. With on-disk locking, VMFS ensures that multiple servers accessing the same storage concurrently do not write simultaneously on the same file. VMFS supports SCSI reservations and atomic test and set locking. SCSI reservations are used on devices that do not support hardware acceleration. The entire volume is locked

during an operation where metadata needs protection. When the operation completes, the lock is released, and access to all ESXi hosts resumes. Atomic test-and-set (ATS) is used in data stores where hardware acceleration is available. Unlike in SCSI reservations, where you lock the entire volume, ATS allows discrete locking per sector.

Figure 3-11 gives an overview of the VMFS architecture.

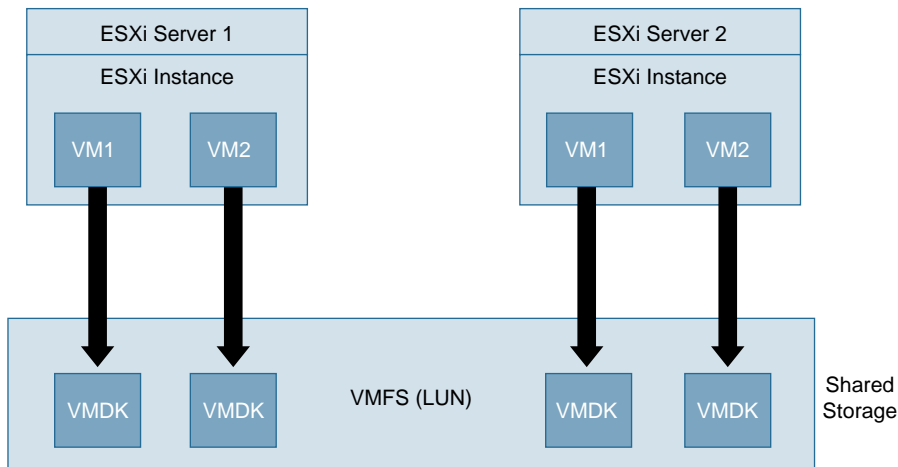


Figure 3-11 *VMFS Architecture*

Management

Management of the hypervisor in ESX was done using a service console. However, as indicated earlier in this chapter, ESXi got rid of the service console, and the management aspect moved from the service console within the kernel to a remote/central location that now enables management using standardized APIs rather than the legacy interactive session service console methodology. In other words, VMware has moved away from the service console operating system that helped interact with the VMkernel in favor of a standardized CIM API model that provides APIs to access the VMkernel.

When a system with ESXi hypervisor boots up, ESXi runs immediately and tries to detect the network by using DHCP. ESXi then spurts out a screen that lets the user configure networking, administrator access password, and the test management network, as shown in Figure 3-12. With the networking option, the user can select the vNICs to assign an IP address and netmask, VLAN, and hostname, for the ESXi host. This enables remote access to the ESXi host using a vSphere client.

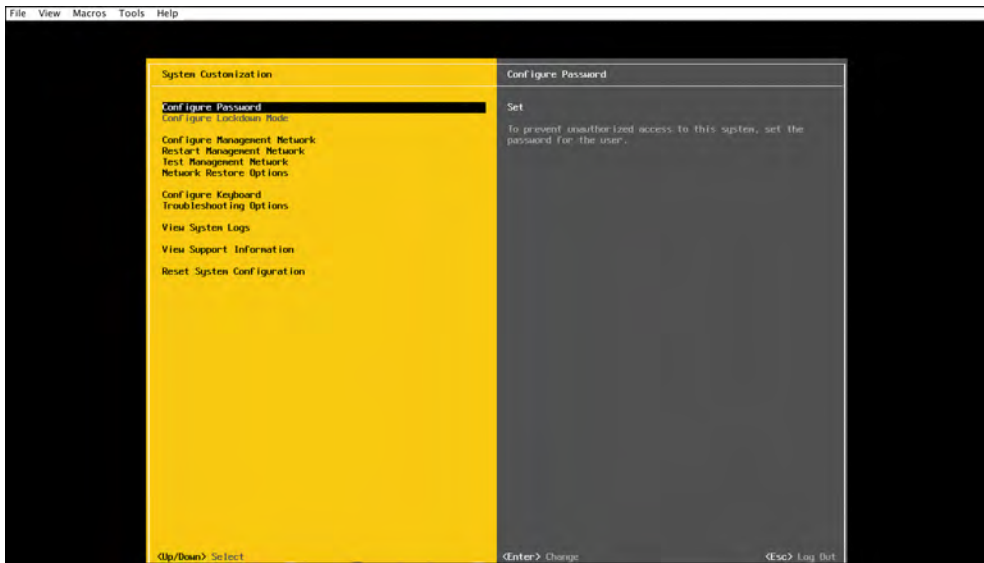


Figure 3-12 *ESXi Opening Screen*

A vSphere client runs on a Windows machine, and VMware now has a web client, too. The vSphere client provides user access to the ESXi host. Through the vSphere client, users can create and manage virtual machines. vSphere clients can let you access an ESXi host directly. However, if you want your vSphere client to manage more than a single instance of ESXi, you need to use vCenter. vCenter is a tool for centrally managing vSphere hosts. With vCenter, you can manage multiple ESXi servers and VMs through a single vSphere client. This makes it possible for a network administrator to take advantage of virtualization features like vMotion, Storage vMotion, fault tolerance, and DRS (Distributed Resource Scheduler). Figure 3-13 shows the vCenter architecture.

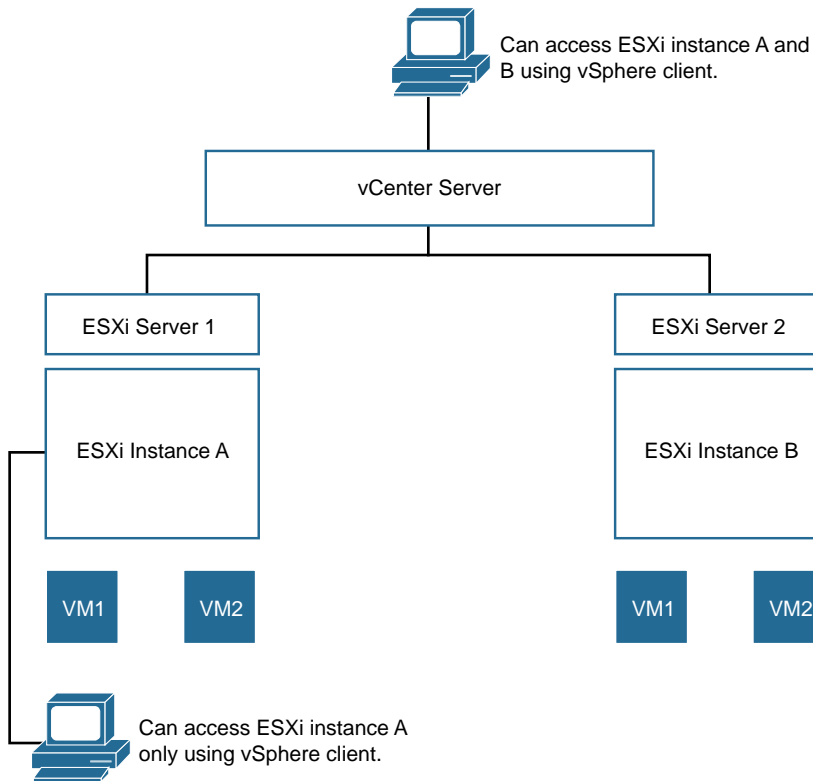


Figure 3-13 *vCenter Architecture*

KVM

Kernel-based Virtual Machine (KVM) is part of the Linux kernel. It is, in fact, the first virtualization solution that has made it to the Linux kernel code. Starting with the 2.6.20 version of the kernel, the KVM module is shipped with the Linux kernel. KVM relies on a virtualization-capable CPU with Intel's Virtualization Technology (VT) or AMD's Secure Virtual Machine (SVM) extension.

KVM was developed initially by an Israeli company, Qumranet (later acquired by Red Hat), which made the KVM code available to the open source community. KVM is a virtualization solution and uses full virtualization to run VMs. KVM's code has deliberately been kept lean and designed leveraging hardware assists. KVM developers wanted to add the bare minimum in terms of components to support full virtualization and wanted it to be an extension of the Linux kernel. With the addition of the KVM module within the Linux kernel, Linux effectively became a hypervisor—capable of hosting virtual machines.

The Linux kernel, sans KVM, is an operating system kernel that runs processes in either guest mode or kernel mode. As described earlier in the chapter, in kernel mode the OS operates with critical data structures or tries to access I/O (also referred to as *controlled resources*). User mode, on the other hand, runs applications. Kernel mode prevents applications in user mode from directly accessing the kernel drivers. KVM achieves this by extending the Linux kernel capability to isolate a process in such a way that it gets its own kernel and user mode. This is called a *guest mode*. Thus a process in guest mode can run its own operating system.

The KVM module uses hardware assists provided by Intel's VT and AMD's SVM processors to execute the guest code directly. KVM treats all its VMs as processes and relies on the scheduler to assign CPUs to the VM. Virtual CPUs (vCPUs) are threads within this VM process. KVM allows the guest user to execute on the physical processor but keeps control of the memory and I/O management. Consider a scenario in which a guest process tries to access a controlled resource. KVM takes control from the guest process and executes the task on the controlled resource on behalf of the guest process. The guest thinks it is running on a real hardware resource. It can do memory paging, segmentation, and interrupt manipulation just as it would if it were running on bare metal. It has its own user and kernel space defined in the guest memory carved for it by KVM.

Figure 3-14 illustrates the KVM modes of operation.

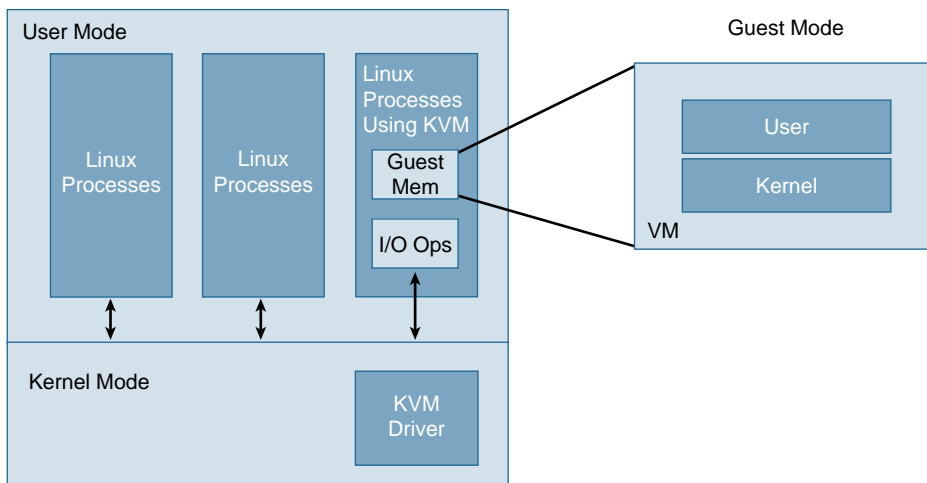


Figure 3-14 KVM Modes of Operation

The guest OS has its own user and kernel modes, and all the user mode functions of the guest OS are executed within this guest-user mode. When the guest OS tries to access the guest-kernel mode, the process exits from the guest-user mode. The Linux/KVM user mode then performs I/O on behalf of this guest.

Architectural Components of KVM/QEMU

Figure 3-15 shows the building blocks of KVM.

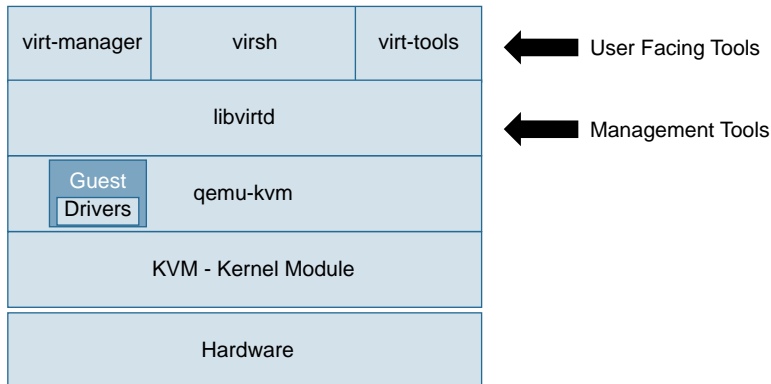


Figure 3-15 *KVM Building Blocks*

The loadable kernel module consists of three files: `kvm.ko`, `kvm_intel.ko` (for Intel processors), and `kvm_amd.ko` (for AMD processors). Linux kernel 2.6.20 and above have these modules included as part of the Linux kernel. `kvm.ko` provides the core virtualization infrastructure, while `kvm_intel.ko` and `kvm_amd.ko` provide the processor-specific module. These KVM modules are the ones that get the virtualization capability to the Linux kernel. This kernel module is responsible for resource management (memory and CPU) for the virtualized environment. KVM natively performs only downstream functions, such as managing the hardware memory and CPU scheduling. It provides a control interface through a set of APIs: `ioctl()` calls for tools (such as QEMU) to emulate hardware virtualization.

Linux Kernel runs the guest process over KVM as a normal Linux process. It does a `malloc()` to allocate memory, and it frees, swaps, and overcommits memory just like a normal process memory.

Each guest maintains its own page table, and so it believes it is doing its own memory management. However, KVM/Linux cannot allow guest operating systems to modify the page table the kernel uses to write to hardware memory. So the host kernel intercepts all the guest memory management unit (MMU) operations and maintains its own *shadow table*, which is a replica of the guest's page table. The host kernel uses this shadow table to write to its physical memory. This way, the guest's virtual memory gets mapped to the host's physical address. However, KVM memory is transparent to the Linux kernel, and Linux treats the memory allocated by KVM no differently than the memory allocated to other Linux processes. Linux tries to swap, free, or replace this memory just as it does for regular process memory. Trouble comes when the guest tries to access the memory Linux just freed. To get around this problem, a feedback mechanism exists with the KVM module that updates the guest of any changes to the shadow tables. `mmu_notifiers` updates the guest about changes to the shadow tables. Only

after the guest has updated its page table is the shadow table updated.

Here is how KVM allocates physical memory to a guest OS:

1. The Guest OS calls for memory. The KVM calls `malloc()`, and a virtual address space is allocated with no physical memory to back it up.
2. When the guest OS process first tries to access this virtual memory, a page fault is generated on the host because there is no physical memory allocated.
3. The kernel calls `do_page_fault()` where the `malloc()` was called and thus allocates physical memory to it. So now the virtual memory has some physical memory to back it up.
4. KVM links the `malloc()`ed virtual address to the physical address allocated on the host and updates `rmaps`.
5. The kernel calls `mmu_notifier` to create an entry for the new page created.
6. The host returns from `page_fault`, and the guest resumes regular operations.

As mentioned earlier, the KVM architectural approach is to utilize most of the Linux kernel functionality and add only what is required for virtualization support. With CPU scheduling, each vCPU (virtual CPU—that the guest OS schedules its processes on) is mapped to a Linux process that uses hardware assistance. This means the vCPU is just like any other Linux process, and Linux uses its CFS (Completely Fair Scheduler) to schedule it on the hardware CPU.

Note As the name suggests, as CFS deploys the algorithm, it attempts to be fair to all processes by providing a fair chance to each process to run on the processor. It keeps an account of time spent by the processor on a process. It tries to prioritize processes that have had less time on the CPU and deprioritize processes that spend a lot of time on the CPU.

Guest Emulator (QEMU)

KVM does not natively spawn a VM. It is just a kernel module that makes the infrastructure ready for you to start a VM. In other words, it executes the low-level kernel functionality but does not create or manage a virtual machine for you. The KVM module makes available within the file system a control interface (`/dev/kvm`) that enables you to control the kernel using `ioctl()` calls. These `ioctl()` calls allow the user to execute code that can enable creation of VMs. A separate tool can then create VMs and use this control interface to schedule memory and CPU for the VMs.

One tool that enables users to emulate virtual machines is Quick Emulator (QEMU). When the VMs spawned by QEMU want to perform an I/O operation, they are intercepted and handled by KVM.

Note Many people compare QEMU and ESXi host, but this is not really fair. QEMU is just a machine emulator and not a hypervisor, as ESXi is. QEMU can help you create and manage VMs when presented with a pool of virtual resources. QEMU, when paired with KVM within the Linux kernel, is a complete package for running and managing VMs. When VMware offers functionalities such as creating and managing VMs, it packages them in ESXi.

QEMU is a free open source tool that emulates the complete hardware of a computer device. QEMU can run on a variety of operating systems and processor architectures. So, unlike VMware, it is not limited to an x86 architecture.

As a guest emulator, QEMU should be able to emulate guest operating systems that run on a physical CPU. In order for QEMU to achieve this, it should be able to do the following:

- Run guest code
- Handle timers
- Process I/O requests from the guest OS
- Monitor command responses

To achieve this, QEMU must be able to execute guest code and schedule its resources in a way that does not pause execution when the I/O response takes a while to complete. There are two architectures available to achieve this:

- **Parallel architecture**—With this architecture, an OS splits the workload into processes or threads that execute simultaneously.
- **Event-driven architecture**—With this architecture, an OS just responds to the events by running one major loop that forks into event handlers.

QEMU uses an architecture that is an amalgamation of the two mentioned here. Its code is event driven, and at the same time, it also uses threads.

The QEMU core is mainly event driven. It is based on an event loop, which dispatches events to handler functions. `main_loop_wait()` is the main event loop in the QEMU core. This is what it does:

- Waits for file descriptors to become readable or writable. You use `qemu_set_fd_handler` to add file descriptor. This registers the file descriptor with the main loop and tells the main loop to wake up whenever certain conditions are met.
- Runs expired timers. You add timers by using `qemu_mod_timer`.
- Runs bottom halves, which are timers that expire immediately. These are used to avoid overflowing the call stack. You add them by using `qemu_bh_schedule`.

When any of these three events occur, `main_loop_wait()` invokes a callback that responds to the event. The callback should be quick to prevent the system from being unresponsive.

For executing guest code, QEMU deploys the following mechanisms:

- **Tiny Code Generator (TCG)**—This mechanism emulates guests by using dynamic binary translation.
- **KVM**—As discussed earlier, the KVM module virtualizes the hardware resources and presents the `/dev/kvm` interface in the Linux file system.

Both TCG and KVM allow the execution control to be given to the guest and allow the guest to execute its code.

QEMU has one thread per vCPU plus a dedicated event loop thread. This is called the `IOTHREAD` model. In the older non-`IOTHREAD` model, one QEMU thread executed the guest code and the event loop. In the new `IOTHREAD` model, each vCPU thread executes the guest code, while the `IOTHREAD` runs the event loop.

Figure 3-16 shows the architecture of a system using KVM and QEMU.

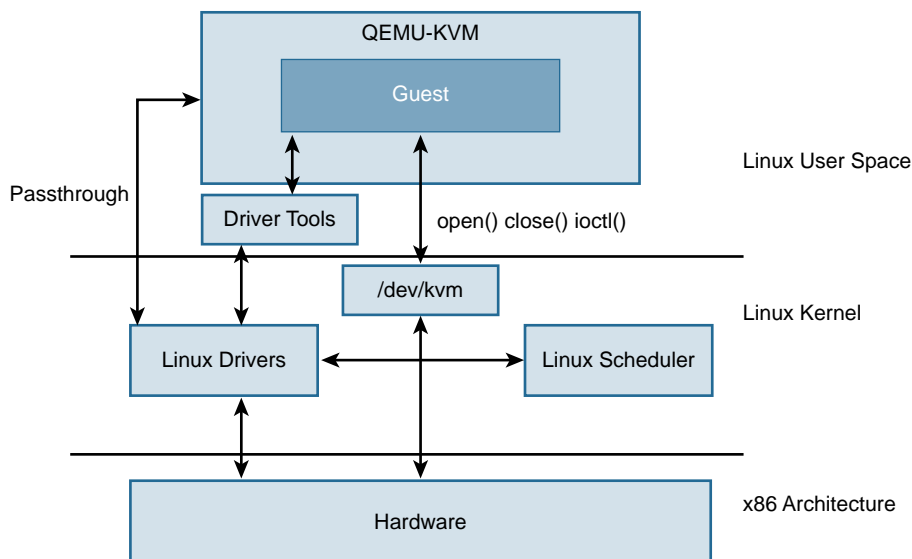


Figure 3-16 QEMU KVM Architecture

Management Daemon (Libvirt)

Management is an important aspect for virtualization. Libvirt is an open source management tool that can be used to manage a virtualized environment. Libvirt, written in C (with bindings to other languages, such as Python), provides an API for managing multiple hypervisors.

Writing and maintaining applications is expensive. Libvirt allows sharing of applications between hypervisors and provides security and remote access, too.

Libvirt is designed to mainly manage virtual machines. It offers the following types of management:

- **VM management**—Libvirt can manage the various life cycle operations of a virtual machine, such as starting, stopping, saving, pausing, moving, and adding/removing CPUs and memory.
- **Access management**—Because Libvirt functionality is available for all machines that run the `libvirtd` daemon, you can use SSH (or any remote login mechanism) for remote access.
- **Network management**—Any host that runs the `libvirtd` daemon can be used to manage physical or virtual interfaces and physical or virtual networks. When the `libvirtd` system daemon is started, a NAT bridge is created. This is called `default` and allows external connectivity. For other network connectivity, you can use the following:
 - A virtual `bridge` that shares data with a physical interface
 - A virtual `network` that enables you to share data with other virtual machines
 - A `macvtap` interface that connects directly to the physical interface on the server on which you host the VM
- **Storage management**—Any host that runs the `libvirtd` daemon can be used to manage various storage types and file formats.

Libvirt management is done mainly using `virsh` and `virt-manager`.

Figure 3-17 shows the Libvirt architecture.

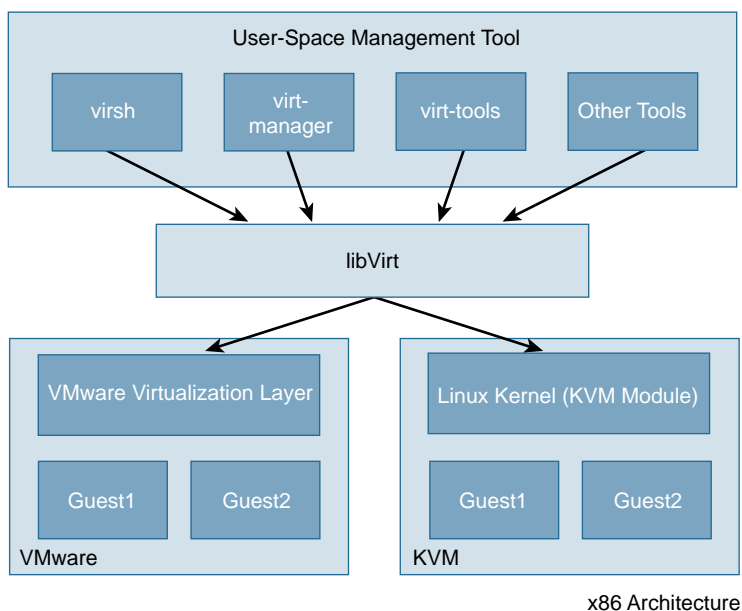


Figure 3-17 *Libvirt Architecture*

User Tools (`virsh`, `virt-manager`)

`virsh` is a piece of code that is used for managing VMs. This command-line tool is very useful for scripting and scaling VM installations. You also get an interactive terminal with `virsh` that can be entered if no commands are passed. Unprivileged users can use `virsh` in read-only mode.

Table 3-1 shows the command-line tools you can use with `virsh` to manage guest VMs.

Table 3-1 *virsh Quick Reference*

| Command | Description |
|----------------------|--|
| <code>help</code> | Prints basic help information. |
| <code>list</code> | Lists all guests. |
| <code>dumpxml</code> | Outputs the XML configuration file for the guest. |
| <code>create</code> | Creates a guest from an XML configuration file and starts the new guest. |
| <code>start</code> | Starts an inactive guest. |
| <code>destroy</code> | Forces a guest to stop. |
| <code>define</code> | Outputs an XML configuration file for a guest. |
| <code>domid</code> | Displays the guest's ID. |
| <code>domuuid</code> | Displays the guest's UUID. |

| Command | Description |
|-----------------------|---|
| <code>dominfo</code> | Displays guest information. |
| <code>domname</code> | Displays the guest's name. |
| <code>domstate</code> | Displays the state of a guest. |
| <code>quit</code> | Quits the interactive terminal. |
| <code>reboot</code> | Reboots a guest. |
| <code>restore</code> | Restores a previously saved guest stored in a file. |
| <code>resume</code> | Resumes a paused guest. |
| <code>save</code> | Saves the present state of a guest to a file. |
| <code>shutdown</code> | Gracefully shuts down a guest. |
| <code>suspend</code> | Pauses a guest. |
| <code>undefine</code> | Deletes all files associated with a guest. |
| <code>migrate</code> | Migrates a guest to another host. |

You can also use `virsh` to manage the resources that are given to the guest or hypervisor with the commands shown in Table 3-2.

Table 3-2 *Commands Used to Manage Resources with `virsh`*

| Command | Description |
|-------------------------------|--|
| <code>Setmem</code> | Sets the allocated memory for a guest. |
| <code>Setmaxmem</code> | Sets the maximum memory limit for the hypervisor. |
| <code>Setvcpus</code> | Changes number of virtual CPUs assigned to a guest. |
| <code>Vcpuinfo</code> | Displays virtual CPU information about a guest. |
| <code>Vcpupin</code> | Controls the virtual CPU affinity of a guest. |
| <code>Domblkstat</code> | Displays block device statistics for a running guest. |
| <code>Domifstat</code> | Displays network interface statistics for a running guest. |
| <code>attach-device</code> | Attaches a device to a guest, using a device definition in an XML file. |
| <code>attach-disk</code> | Attaches a new disk device to a guest. |
| <code>attach-interface</code> | Attaches a new network interface to a guest. |
| <code>detach-device</code> | Detaches a device from a guest and takes the same kind of XML descriptions as the command <code>attach-device</code> . |
| <code>detach-disk</code> | Detaches a disk device from a guest. |
| <code>detach-interface</code> | Detaches a network interface from a guest. |

With these commands you can manage the guest effectively and automate a lot of guest bring-up and configuration sequences.

`virt-manager` provides a graphical way to manage VMs and hypervisors. It does the same things as `virsh` but via a user-friendly graphical interface.

Hyper-V

Hyper-V, formerly known as Windows Server Virtualization, can create virtual machines on the x86 platform. In October 2008, Hyper-V Server 2008 was released, and since then it's been used as a hypervisor platform for other members of the Windows Server family.

Hyper-V is a type 1 microkernel hypervisor that resides directly on the hardware. The microkernel architecture optimizes performance and reduces adoptability issues with the underlying hardware.

The architecture uses a parent VM that hosts the drivers. The guest operating system interfaces with the parent partition to access hardware resources' memory, CPU, storage, and so on. The guest operating system works within the privileged boundary. Figure 3-18 gives a high-level overview of the Hyper-V architecture.

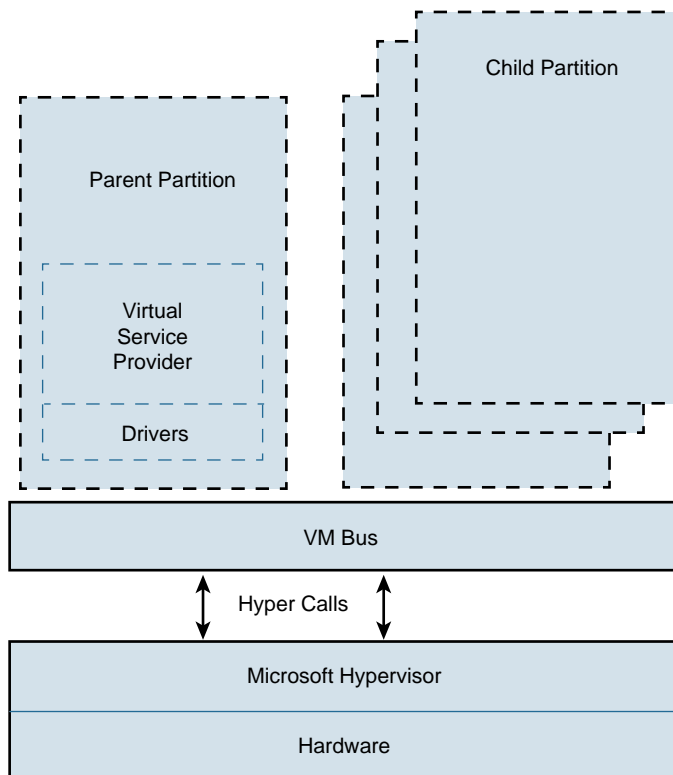


Figure 3-18 *Hyper-V Architecture High-Level View*

On the top of the hypervisor is one parent partition and one or more child partitions. This partitioning creates virtual isolation within the hypervisor for physical memory address space and virtual processors.

The child partition can host a guest operating system or system functions for Microsoft Windows Server. For example, when virtual Hyper-V stack management gets installed in the parent partition, the subsidiary Windows Server functionality is loaded in the child partition. Each virtual machine has its own security boundary. Microsoft refers to this as an operating system environment (OSE) that defines the component of a virtual machine. The VM has its own identity, computer account, IP address, and network resources. Child partitions have only the virtual view of the hardware resources. The child partition sends a request to the virtual devices, which gets redirected to the hypervisor in the parent partition that handles this request.

The parent partition has access to hardware devices and controls the resources used by the child partition. The child partition accesses the hardware resource using the drivers in the parent partition space. The parent partition also acts as the broker among the multiple child partitions and the hardware for accessing the resources. All data and instructions between the parent and child partition go through the virtual machine bus. The architecture allows the user to leverage plugin devices, which in turn allow direct communication between parent and child partitions.

Xen

Xen is another type 1 hypervisor that supports para-virtualization. Xen originated as a college research project at the University of Cambridge. Ian Pratt, who was a lecturer in Cambridge, led this project and later cofounded XenSource, Inc. First available in 2004, Xen was originally supported by XenSource, Inc. Eventually, Xen was moved under the Linux Foundation as a collaborative project.

Starting with Xen 3.0, all guest VMs can run their own kernels and take advantage of para-virtualization, which removes the need to emulate virtual hardware resources, makes the guest aware of the hypervisor, and enables access to the hardware resources for I/O efficiency. Instead of the guest spending time and extra cycles performing tasks to get resources from the virtual environment, these guests can use the hooks of para-virtualization to allow guest and host to communicate and acknowledge these tasks.

Figure 3-19 gives a high-level architectural overview of the XEN hypervisor.

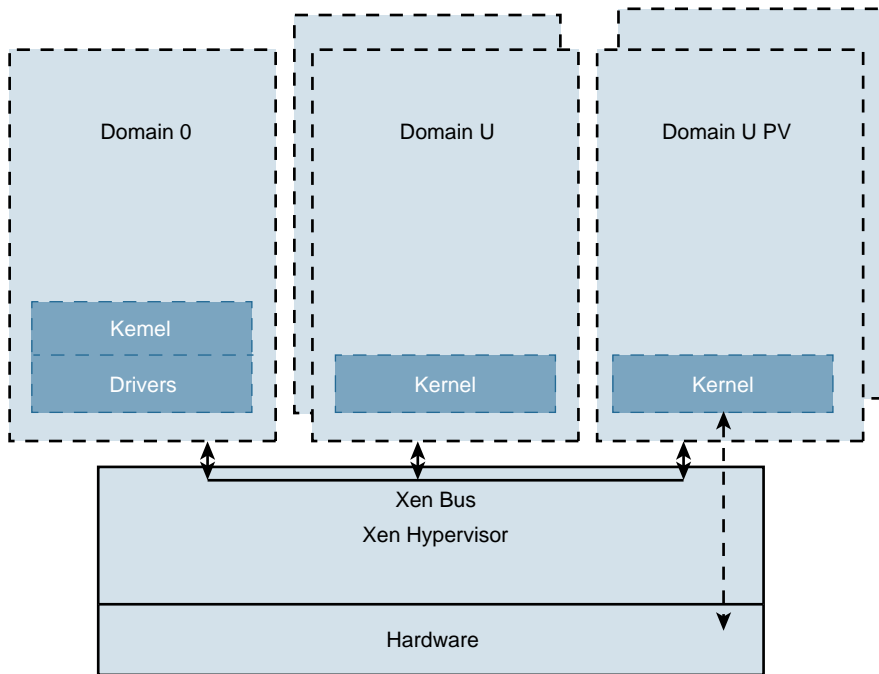


Figure 3-19 *Xen Architecture High-Level View*

These are the key components of XEN architecture:

- **Xen hypervisor**—The hypervisor sits directly on the hardware. Its key functions are CPU scheduling and memory partitioning. The VM that requires abstraction of hardware and resources can leverage the hypervisor for control of shared resources. The hypervisor does not have knowledge of networking, storage, or the common I/O.
- **Domain 0** —This VM resides on the Xen hypervisor and owns the right to access physical I/O resources as well as interact with other guest VMs that need access to these resources. Domain 0 has to be set up before any other guest VMs are spawned.
- **Domain U and Domain U PV** —Domain U is an unprivileged guest. Domain 0 uses drivers for the network and storage located in Domain 0. The Domain U guest must communicate via the Xen hypervisor with Domain 0 to accomplish a network or disk request. Domain U PV is capable of para-virtualization. Guests have direct access to the hardware resources for disk and network access. Domain U PV and Domain 0 can access the shared network and disk resources. This is done using an event channel that exists between Domain 0 and Domain U PV. This event channel allows the domains to communicate via asynchronous communication, using interdomain interrupts in the Xen hypervisor. The Xen hypervisor sends an interrupt to Domain 0 and allows Domain U PV to access the shared memory resources. The Xen PCI passthrough feature is used for the Domain U PV to access other non-disk hardware resources directly.

Summary

Now that you've read this chapter, you should have a basic understanding of operating systems and types of hypervisors. This chapter reviews the details of KVM and ESXi and provides an overview of Hyper-V and Xen. Understanding the hypervisor types is important for the deployment of CSR 1000V.

Table 3-3 provides a summary of the hypervisor types you need to know about.

Table 3-3 *Hypervisor Types*

| | Type of Hypervisor | Open Source? | Para-virtualization Support? |
|---------|---------------------------|---------------------|-------------------------------------|
| ESXi | Type 1 | No | Yes |
| KVM | Type 1 | Yes | Yes |
| Hyper-V | Type 1 | No | Yes |
| Xen | Type 1 | Yes | Yes |

All of these hypervisors support the key features, so the use cases depend on the environment and operational support criteria.

CSR 1000V Software Architecture

This chapter describes the software design of the CSR 1000V and details the data plane design. It also illustrates the software implementation and packet flow within the CSR 1000V, as well as how to bring up the CSR 1000V.

System Design

CSR 1000V is a virtualized software router that runs the IOS XE operating system. IOS XE uses Linux as the kernel, whereas the IOS daemon (IOSd) runs as a Linux process providing the core set of IOS features and functionality. IOS XE provides a native Linux infrastructure for distributing the control plane forwarding state into an accelerated data path. The control and data planes in IOS XE are separated into different processes, and the infrastructure to communicate between these processes supports distribution and concurrent processing. In addition, IOS XE offers inherent multicore capabilities, allowing you to increase performance by scaling the number of processors. It also provides infrastructure services for hosting applications outside IOSd.

Originally, IOS XE was designed to run on a system with redundant hardware, which supports physical separation of the control and data plane units. This design is implemented in the ASR 1006 and ASR 1004 series routers. The original ASR 1000 family hardware architecture consisted of the following main elements:

- Chassis
- Route processor (RP)
- Embedded service processor (ESP)
- SPA interface processors (SIP)

The RP is the control plane, whereas the ESP is the data plane. In an ASR 1006 and ASR 1004, the RP and ESP processes have separate kernels and run on different sets of hardware. ASR 1000 was designed for high availability (HA). The ASR 1006 is a fully

hardware redundant version of the ASR, and its RP and ESP are physically backed up by a standby unit. IOSd runs on the RP (as do the majority of the XE processes), and the RP is backed up by another physical card with its own IOSd process. The ASR 1004 and fixed ASR 1000s (ASR 1001-X and ASR 1002-X) do not have physical redundancy of the RP and ESP.

In the hardware-based routing platform for IOS XE, the data plane processing runs outside the IOSd process in a separate data plane engine via custom ASIC: QuantumFlow Processor (QFP). This architecture creates an important framework for the software design. Because these cards each have independent processors, the system disperses many elements of software and runs them independently on the different processors.

Tip The ASR 1000 platform first introduced IOS XE. Multiple products run IOS XE, including the following:

ASR 1000 family:

- ASR 1001-X
- ASR 1002-X
- ASR 1004
- ASR 1006
- ASR 1006-X
- ASR 1009-X
- ASR 1013

ASR 900 family:

- ASR 903

ISR family:

- ISR 4321
- ISR 4331
- ISR 4351
- ISR 4431
- ISR 4451-X

IOS XE retains the look and feel of IOS. However, because IOS runs as a Linux process, it enables the platform-independent code to reside inside the IOSd process running on the Linux kernel. By moving the platform-dependent code (drivers) outside the IOSd process, it makes IOS XE a very efficient software delivery model. Different platforms write their drivers and leverage the existing feature-rich control plane code from IOSd.

Multiple platforms run IOS XE. However, when understanding CSR 1000V architecture in this chapter, ASR 1000 is used as a hardware example because it was the first platform to run IOS XE.

As the need for smaller form factor ASRs arose, a one rack unit (RU) ASR 1000 was conceptualized and developed: ASR 1001. The ASR 1001 is a 64-bit architecture in which all processes (RP, SIP, and ESP) are controlled by a single CPU. The SPA interface complex, forwarding engine complex, and IOS XE middleware all access the same Linux kernel. This is achieved by mapping the RP, ESP, and SIP domains into logical process groups. The RP's process domain includes IOSd, a chassis manager process and forwarding manager. The ESP process domain includes the chassis manager process, QFP client/driver process, and forwarding manager.

The architecture diagram in Figure 4-1 provides a high-level overview of the major components.

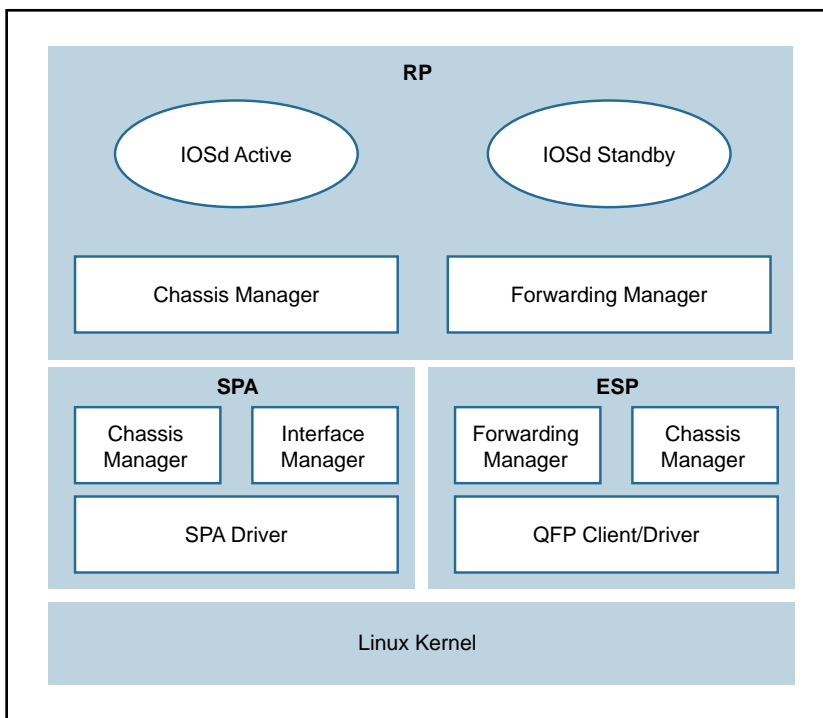


Figure 4-1 ASR 1001 Platform Logical Architecture

The details on grouping of the components are as follows:

- **RP**—RP mainly contains the IOS daemon (IOSd), the forwarding manager for RP (FMAN-RP), the chassis manager for RP (CMAN-RP), the kernel, and bootstrap utilities.

- **ESP (forwarding plane)**—ESP contains FMAN-FP and CMAN-FP, as well as QFP microcode and data plane drivers and crypto offload ASIC for handling hardware assist encryption.
- **SIP/SPA**—SIP/SPA houses the I/O interface for the chassis. It has its own CMAN and kernel process to handle the discovery, bootstrapping, and initialization of the physical interfaces.

Virtualizing the ASR 1001 into the CSR 1000V

There are a lot of commonalities between the system architectures of the CSR 1000V and the ASR 1001, and there are some differences as well. The CSR 1000V is essentially an ASR 1001 without the hardware. The following measures brought the ASR 1001 into the software-based design of the CSR 1000V:

- All the inter-unit communication with the SIP/CC was removed.
- The entire SIP/SPA interface complex was eliminated.
- The kernel utilities have been shared across the RP and ESP software complexes.
- The kernel utilities use the virtualized resources presented to it by the hypervisor.

The CSR is basically the ASR 1000 design stripped of its hardware components. When you compare the two designs, you find that the data path implementation is very different. This is because the ASR 1001 has a physical processor (the QFP) for running data path forwarding. In a CSR, the IOS XE data path is implemented as a Linux process.

The CSR 1000V is meant to leverage as much of the ASR 1001 architecture as possible. There are places in the CSR 1000V system where software emulation for hardware-specific requirements is needed. In general, the software architecture is kept the same, using the same grouping approach as for the hardware components. One of the major engineering efforts has been focused on migrating the QFP custom ASIC network processor capabilities onto general-purpose x86 CPU architectures and providing the distributed data path implementation for IOS XE. This effort creates a unique opportunity for Cisco to package this high-performance and feature-rich technology into the CSR 1000V. Figure 4-2 shows the high-level architecture of the CSR 1000V.

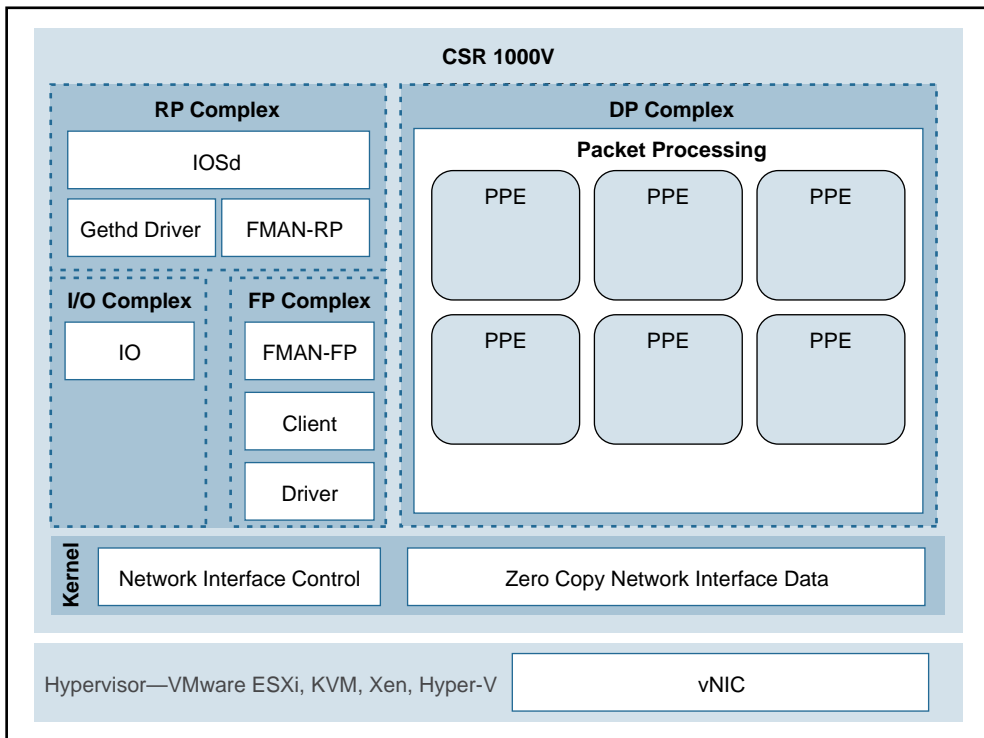


Figure 4-2 CSR 1000V High-Level Architecture

CSR 1000V Initialization Process

This section examines the initialization of a CSR 1000V running on a type 1 hypervisor. Refer to Chapter 2, “Software Evolution of the CSR 1000,” for details on the IOSd process running on the control plane.

When a CSR boots up as a virtual machine, interfaces are discovered by parsing the contents of `/proc/net/dev` on the Linux kernel. The `gethd` (Guest Ethernet Management Daemon) process performs the port enumeration at startup and then passes the interface inventory to the guest Ethernet driver within the IOS complex. The IOSd `gethd` driver then instantiates the Ethernet interfaces. This is how the I/O interfaces provided by the virtual NIC are managed by IOS.

The `gethd` process manages the interfaces on the CSR VM. It takes care of addition, removal, configuration, states, and statistics of the Ethernet interfaces on the CSR VM.

Figure 4-3 illustrates the CSR 1000V initialization sequence.

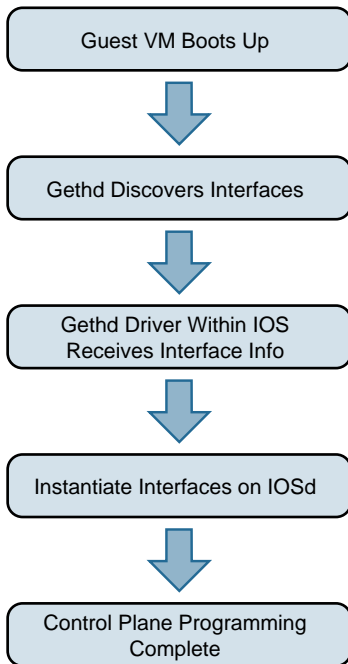


Figure 4-3 *CSR 1000V Initialization Sequence*

`gethd` is an important process that handles a variety of interface management functions, including interface removal/addition. It is an important part of the virtualized I/O used in CSR.

CSR 1000V Data Plane Architecture

Originally, IOS XE QFP data plane design consisted of four components: client, driver, QFP microcode (uCode), and crypto assist ASIC. Different ASR 1000 platforms package these components differently, but in general the four components are the same across platforms. CSR 1000V leverages the same client, driver, and uCode to support a multithread-capable packet processing data plane, with the exception of the crypto assist ASIC.

Figure 4-4 illustrates the CSR 1000V data plane architecture. The HW threads mentioned in the figure are packet processing engine (PPE) threads. The terms *HW* and *PPE* can be used interchangeably.

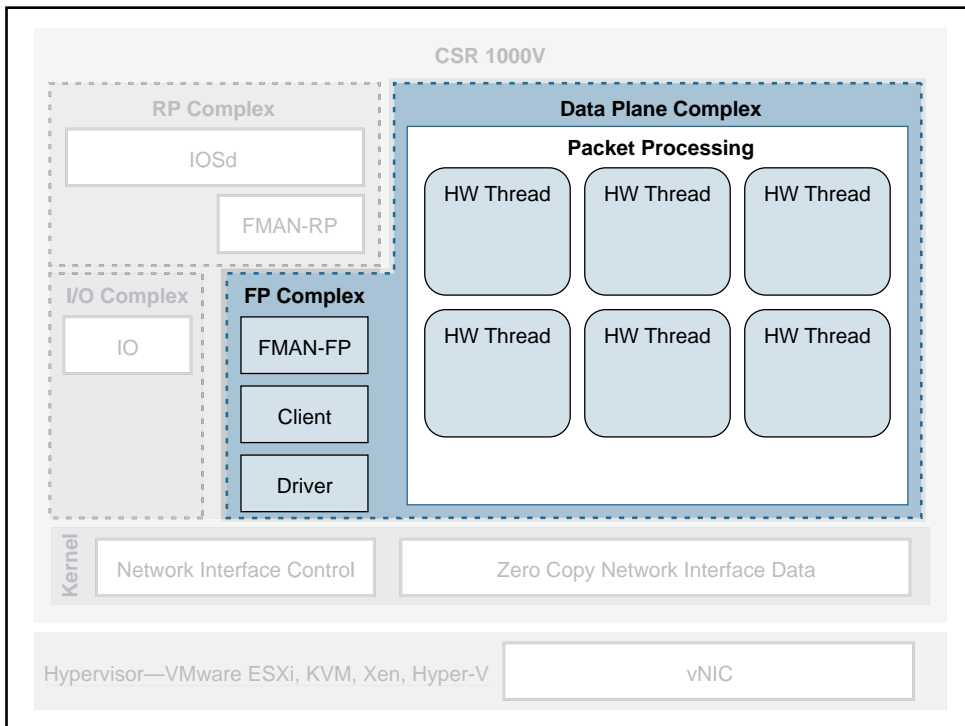


Figure 4-4 CSR 1000V Data Plane Architecture

The following is an overview of the three main components that make up the packet-processing data plane for CSR:

- Client**—The Client is software that ties together the control plane and the data plane. It is a collection of software modules that transform control plane information into various data plane forwarding databases and data structure updates. It is also responsible for updating the control plane with statistics from the data plane. It allocates and manages the resources of the uCode, including data structures in resource memory. The QFP Client is also responsible for restarting the QFP process in the event of failure. The Client provides a platform API layer that logically sits between IOSd and the uCode implementing the corresponding features. The Client API is called from FMAN-FP and then communicates with the uCode via both Interprocess Communicator (IPC) and shared memory interfaces provided by the Driver. Within the Client, feature processing support can be broken down into functional blocks known as Execution Agents (EA) and Resource Managers (RM). RMs are responsible for managing physical and logical objects, which are shared resources. An example of a physical object manager is the TCAM-RM, which manages allocation of TCAM resources, and an example of a logical object manager is the UIDB-RM, which manages the micro Interface Descriptor Block (uIDB) objects

used to represent various forms of interfaces. The data plane (uCode) uses uIDB objects to see the logical interfaces.

- **Driver**—The Driver is a software layer that enables software components to communicate with the hardware. It glues the software components to the hardware. The Driver is made up of libraries, processes, and infrastructure that are responsible for initialization, access, error detection, and error recovery. The Driver has hardware abstraction layering known as the Device Object (`devobj`) Model that allows it to support different QFP ASICs. Below the `devobj` API are implementations of various emulation and adaptation layers. In addition to the emulation and adaptation layers required to support the RMs listed in the Client section, the Driver is also responsible for coordinating memory access and IPC messaging between various QFP control plane software components and the QFP data plane packet processing uCode. The driver is completely segregated from the IOS code in an XE architecture, and this makes XE a very robust and flexible software architecture that offers complete separation of the control and data planes.
- **QFP uCode (packet processing)** —The uCode is where all the feature packet processing occurs. The uCode runs as a single process in the same VM/container as the Client and the Driver processes. IOSd initiates a packet process request through FMAN-FP. This request is then driven by the Client and the Driver interacting with the uCode to control the PPE behavior. The QFP uCode is broken up into four main components: Feature Code, Infrastructure, Platform Abstraction Layer (PAL), and Hardware Abstraction Layer (HAL). The PAL and HAL are essentially glue for the portability of software features to different hardware platforms. Originally, the PAL and HAL were designed for Cisco forwarding ASICs, such as QFP. In order for uCode software to run on top of x86 in a Linux environment, a new PAL layer is needed to support the specifics of the CSR 1000V platform. In addition, a new HAL is introduced for running QFP software on top of x86 in a Linux environment.

The intention is for the CSR 1000V data plane to leverage as much of the existing QFP code base as possible to produce a full-featured software data plane capable of leveraging the processing capacity and virtualization capabilities of modern multicore CPUs. One way to minimize changes to the existing QFP software code base is to emulate QFP hardware ASIC in such a way that the existing Client, Driver, and QFP uCode are not aware that they are running on a non-QFP platform. However, due to the complexity of QFP hardware and the differences in platform requirements, a pure emulation is impractical. There are some cases where we choose to emulate hardware because doing so is the straightforward approach for code leverage. In other cases, it is best to replace the corresponding functionality with an implementation that is compatible at an API level but may be a completely different algorithmic implementation.

CSR 1000V Software Crypto Engine

Cisco router platforms are designed to run IOS with hardware acceleration for crypto operations. Like other ASR 1000 platforms, the ASR 1001 includes a crypto acceleration engine on board to deliver crypto offload and to increase encryption performance. In this environment, the main processor performing the data path processing is offloaded from the computing-intensive crypto operations. Once the crypto offload engine completes the encrypt/decrypt operation, it generates an interrupt to indicate that the packet should be reinserted back into the forwarding path.

The CSR 1000V runs completely on general-purpose CPUs without an offload engine; therefore, the software implementation of the IPsec/crypto feature path is needed to support the encryption function. To that end, the CSR 1000V includes a software crypto engine that uses low-level cryptographic operations for encrypting and decrypting traffic. The software crypto engine is presented to the IOS as a slower crypto engine. One thing to note is the software crypto engine runs as an independent process within the CSR 1000V, and it therefore may run as a parallel process in a multicore environment. To improve the crypto performance of the CSR 1000V software router, the crypto data path is implemented to take advantage of the latest Advanced Encryption Standard (AES) crypto instruction set from Intel (AES-NI) for encryption/decryption operations.

The newer Intel processors, such as the Xeon Westmere-EP family and mobile Sandy Bridge family, provide instruction sets for enhancing Advanced Encryption Standard (AES-NI) cryptographic operations performance. These instructions are included in the CSR 1000V crypto library, along with other cryptographic and hash algorithms for low-level crypto operations. The crypto library is used by the software crypto engine as well as by other subsystems within IOS that require cryptographic operations. The inclusion of Intel's crypto instruction set allows the CSR 1000V to take advantage of the latest Intel CPUs for encryption and decryption operations in the data path.

Life of a Packet on a CSR 1000V: The Data Plane

Before we get into the details of packet flow for the CSR 1000V, it is important to understand the drivers that make it possible for the CSR VM to talk to physical devices and other software modules. These drivers act as software glue, relaying a packet to and from the physical wire. We have touched on the different hypervisors that enable the CSR VM to work on various x86 architectures. Here we discuss packet flow to and from a CSR VM.

Figure 4-5 shows the virtualization layers of a CSR 1000V VM.

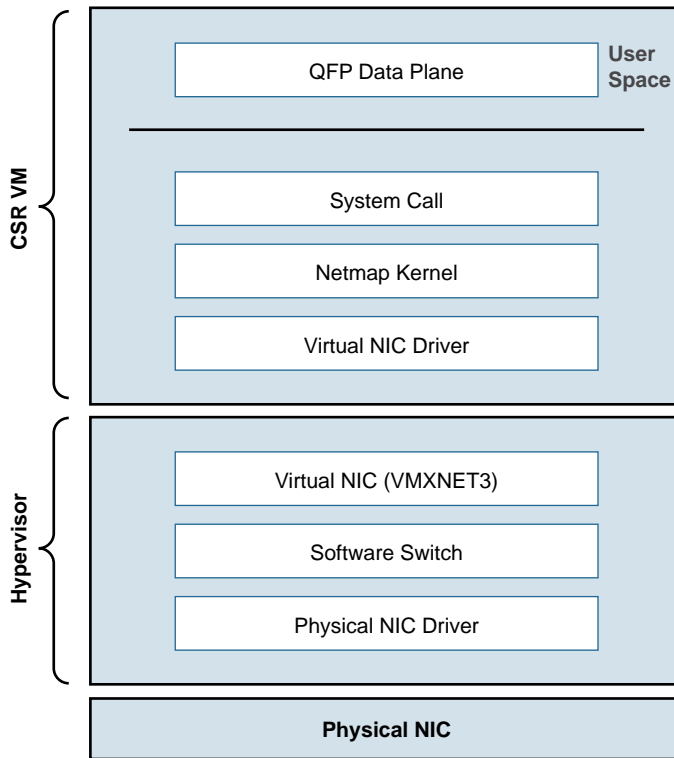


Figure 4-5 CSR VM Layers

From Figure 4-5, you can see that the hypervisor presents a virtual NIC to its guest VM by using a driver. This driver can either be a para-virtualized driver (for example, VMXNET3) or a real/emulated driver (for example, e1000). Para-virtualized drivers are native to hypervisors and perform much better than emulated drivers such as the e1000. Hypervisors support emulated drivers because they are required for full virtualization. Recall from Chapter 1, “Introduction to Cloud,” that in full virtualization, guest operating systems do not require any support from the hypervisor kernel and run as though on real hardware. Therefore, support for emulated drivers is required. However, the performance of emulated drivers is much lower than that of para-virtualized drivers. The CSR VM supports para-virtualized drivers only.

Netmap I/O

Netmap is an open-source I/O infrastructure package that enables the CSR VM to get rid of the multiple software layers in the traditional Linux networking stack I/O model. This results in faster I/O. Understanding the Netmap I/O model will help you better understand packet flow to and from a CSR VM. This section provides an overview of the Netmap I/O model and compares it with a Linux I/O model. It is important to understand the I/O model before drilling down to packet flow.

Netmap is designed to strip down software layers and get the frame from the wire to the data plane process in user space as quickly as possible. Netmap achieves this through the four building blocks of its I/O architecture:

- **Thin I/O stack**—Netmap bypasses the Linux networking stack to reduce overhead. Since the CSR data plane runs in the user space, when it wants an I/O architecture to deliver receive (Rx) frames from the NIC to the user space (data plane) and transmit (Tx) frames from the data plane to the NIC, it leverages Netmap’s thin I/O stack.
- **Zero copy**—Netmap maps all memory from rings (pool of memory buffers) in a way that makes them directly accessible in the data plane (user space). Hence there is no copy involved in getting the information to the user space. Preventing a copy operation saves a lot of time in an I/O model, and Netmap’s zero-copy model is very effective at increasing performance compared to a traditional Linux I/O model.
- **Simple synchronization**—The synchronization mechanism in Netmap is extremely simple. When you have the Rx packets on the ring, Netmap updates the count of new frames on the ring and wakes up threads that are sleeping to process the frames. On the Tx side, the write cursor is updated as a signal to announce the arrival of new frames on the Tx ring. Netmap then flushes the Tx ring.
- **Minimal ring manipulation**—In the Netmap I/O architecture, the ring is sized such that the producer accesses the ring from the head end, while the consumer accesses it from the tail. (*Producer* and *consumer* are terms associated with the process that tries to initiate the I/O process [producer] and a process that gets affected in trying to serve the producer [consumer].) The access to the ring is allowed simultaneously for the producer and the consumer. In a regular Linux I/O scenario, you would have to wait for the host to fill up the ring with pointers to buffers. When the ring is being serviced, Linux detaches the buffers from the ring and then replenishes the ring with new pointers.

An overview of the layers of software involved in building a CSR 1000V VM is illustrated previously in Figure 4-5. Figure 4-6 compares the Linux I/O model with the Netmap I/O model.

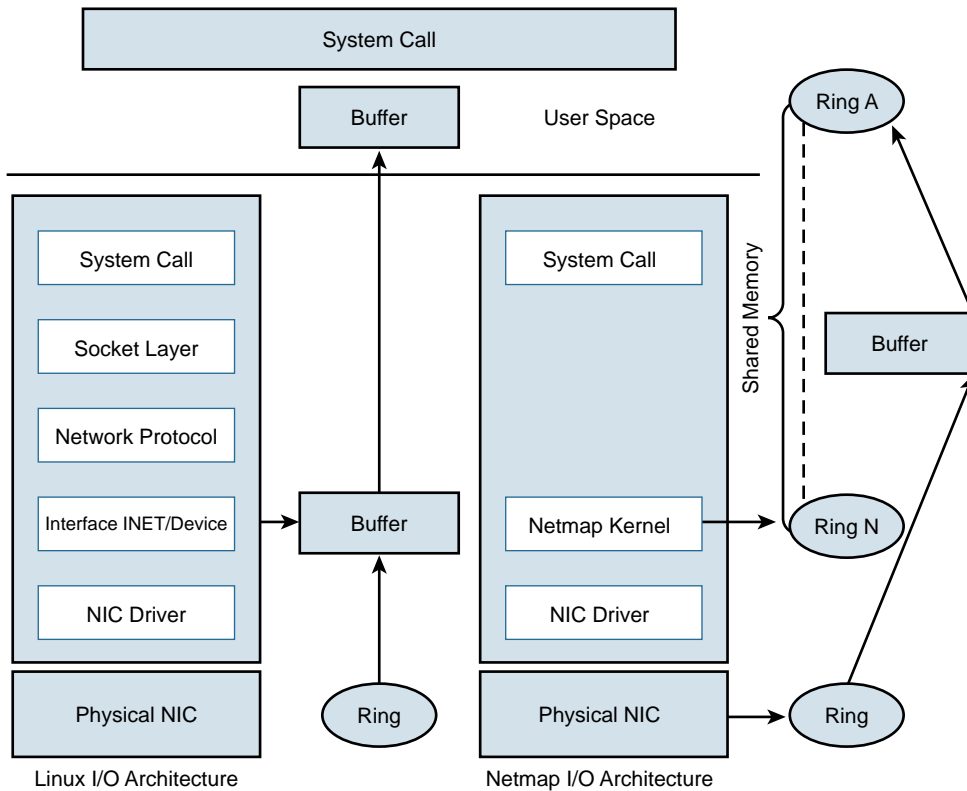


Figure 4-6 *Linux Versus Netmap I/O Comparison*

Packet Flow

There are three major data plane components:

- Rx thread
- Tx thread
- HQF (Hierarchical Queuing Framework) thread

All these components run on a single process within the QFP process umbrella. Multiple PPE threads serve requests within this QFP process. The following sections discuss the flow.

Device Initialization Flow

The following events take place to get the NIC (or vNIC, in a para-virtualized environment) ready for operation:

1. During boot-up, the platform code within IOSd discovers all Linux network interfaces. The platform code then maps these Linux interfaces—`eth0`, `eth1`, and so on—to `Gig0`, `Gig1`, and so on. After talking to the kernel, platform code sets up the interface state (up or down), sets the MTU, sets the ring size, and sets the MAC address.
2. The FMAN process creates the FMAN interfaces and then reaches out to the QFP client process to initialize the data-plane interface.
3. After the QFP process receives the initialization message from the Client process to create an interface, the QFP process then initializes an interface called micro-interface descriptor block (uIDB) in the data plane.
4. After the uIDB is created in the QFP process, the FMAN process binds this uIDB to the network interface name.
5. The component of the data-plane process responsible for interacting with the kernel now has to make sure that the interface created with the QFP process is registered and enabled within the Netmap component of the kernel.
6. With the new interfaces registered, the Netmap component communicates with the virtual NIC driver to initialize the physical NIC.
7. The vNIC driver opens the NIC, initiates the rings, and makes the NIC ready for operation.

TX Flow

The following events take place when there is a packet to be transmitted (Tx) by the CSR onto the wire:

1. The HQF thread detects that there are packets to be sent.
2. The HQF thread checks congestion on the transmit interface and checks the interface states.
3. If the transmit interface is not congested, HQF sends the frame. HQF can also wait to accumulate more frames, batch them, and then send them out.
4. The platform code locates the next available slot in the Tx ring and copies the frame from the source buffer into the Netmap buffer for transmission.
5. The platform code flushes the Tx ring.
6. Netmap forwards the flushed frames to the vNIC driver.
7. The vNIC driver initializes the NIC Tx slots.
8. The vNIC driver writes onto the Tx registers.
9. The vNIC driver cleans up the Tx ring of done slots.
10. The vNIC sends the frame on the wire and generates a notification on completion.

RX Flow

The following events occur whenever a CSR receives a packet to be processed:

1. The Rx thread (the thread that receives frames from the QFP process) issues a poll system call to wait for the new Rx frames.
2. When a new frame arrives, the NIC (or vNIC, in this case) accesses the vNIC Rx ring to get a pointer to the next Netmap buffers.
3. The vNIC puts the frame onto the next Netmap buffers.
4. The vNIC generates an Rx interrupt.
5. The Netmap Rx interrupt service routine runs the Rx threads.
6. The vNIC driver finds the new frame and creates memory buffers for it.
7. The Rx thread pushes the frame to the PPE thread for processing.

Figure 4-7 illustrates packet flow between different XE processes.

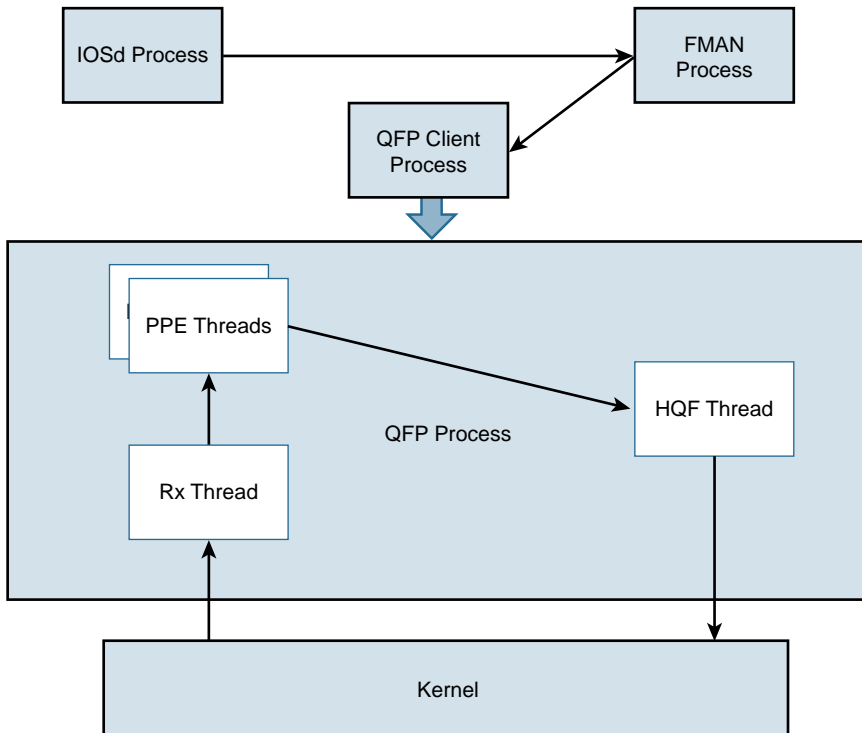


Figure 4-7 Flowchart for Packet Flow

Unicast Traffic Packet Flow

The Tx and Rx flows in Figure 4-7 detail how a packet is transmitted from the NIC (or vNIC, in a para-virtualized driver) to the QFP process. Now we can look at how the QFP process handles the packet after it gets it. The following steps examine a unicast IPv4 packet flow:

1. The QFP process receives the frame from the Netmap Rx and stores it in Global Packet Memory (GPM).
2. The Dispatcher copies the packet header from the GPM and looks for free PPE to assign. The packet remains in the GPM while it is being processed by the PPEs.
3. The Dispatcher assigns a free PPE thread to process the feature on the packet.
4. PPE threads process the packet and gather the packets. The *gather* process copies the packets into B4Q memory and sends the HQF thread a notification that there is a new packet in the B4Q memory.
5. HQF sends the packet by copying it from B4Q into the Netmap Tx ring, and then releases the B4Q buffer.
6. The Ethernet driver sends the frame and frees the Tx ring once the packet has been sent out.
7. Multicast IPsec packets are recycled from the HQF thread back to the in/out processing of the PPE threads.

Figure 4-8 illustrates the packet flow in the QFP process.

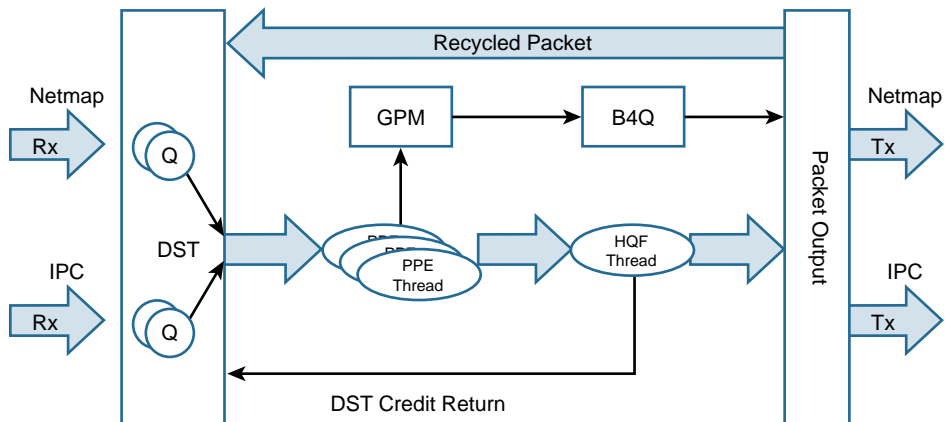


Figure 4-8 CSR 1000V Packet Flow in the QFP Process

Installing the CSR 1000V on a VMware Hypervisor

The process for installing the CSR 1000V on a VMware hypervisor has two phases:

1. Bring up the VM with the CSR 1000V on ESXi.
2. Connect the VNIC with the CSR 1000V.

These phases can be subdivided into the step-by-step procedures described in the following sections. To learn about automated provisioning using the BDEO (build, deploy, execute OVF), see Chapter 7, “CSR in the SDN Framework.”

The following steps assume ESXi is already installed. Please refer to the VMware ESXi installation guide for setting up the ESXi if it is not already installed.

Bringing Up the VM with the CSR 1000V on ESXi

Assuming ESXi is already installed, you can now follow these steps in the first phase of installing the CSR 1000V:

Step 1. Deploy the OVF template:

1. Download the OVF template from software.cisco.com and select CSR 1000V software.
2. Log on to the vSphere client, as shown in Figure 4-9.

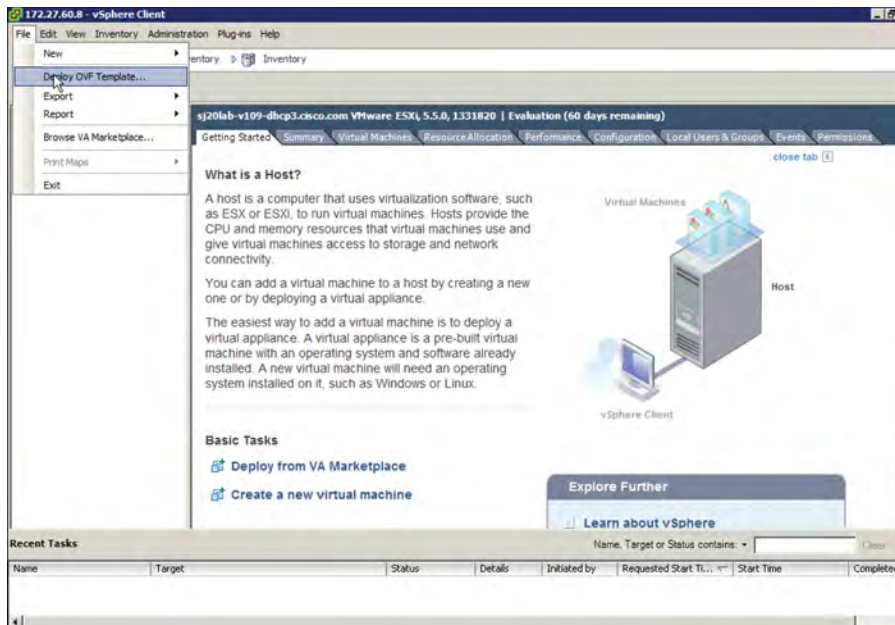


Figure 4-9 Installing the OVF Template for the CSR 1000V

3. Upload the CSR OVF file you downloaded from cisco.com as shown in Figure 4-9.

4. Select File, Deploy OVF Template, as shown in Figure 4-9.

Step 2. Upload the CSR OVF file as shown in Figure 4-10.

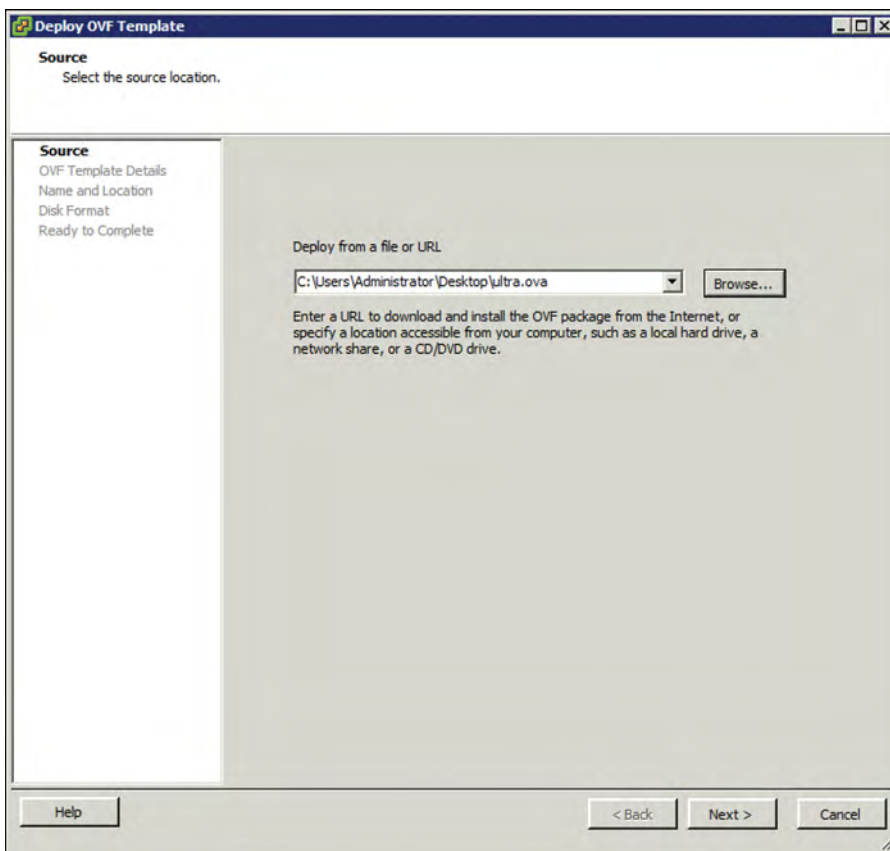


Figure 4-10 *Deploying the OVF Template: Selecting the Source*

Step 3. When the OVA upload is done, verify the OVF template details on the screen shown in Figure 4-11.

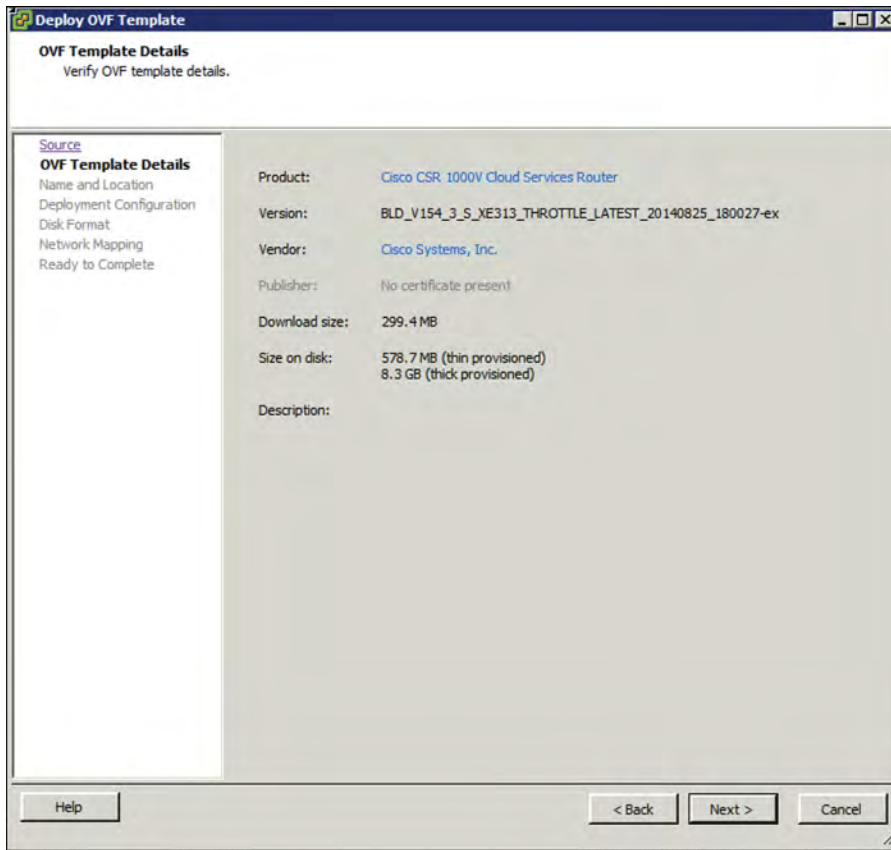


Figure 4-11 *Deploying the OVF Template: Verifying the Template Details*

The release information, product, size, and so on are received from the metadata. Follow the directions for creating the VM.

Complete the following deployment configuration, disk formatting, and network mapping screens, as shown in Figures 4-12 through 4-16:

1. As shown in Figure 4-12, select the hardware profile: Small, Medium, or Large vCPU and RAM, based on the deployment considerations. Refer to the hypervisor documentation for the exact small, medium, and large VM configurations. (You can change this configuration for memory even after the CSR 1000V is brought up.)

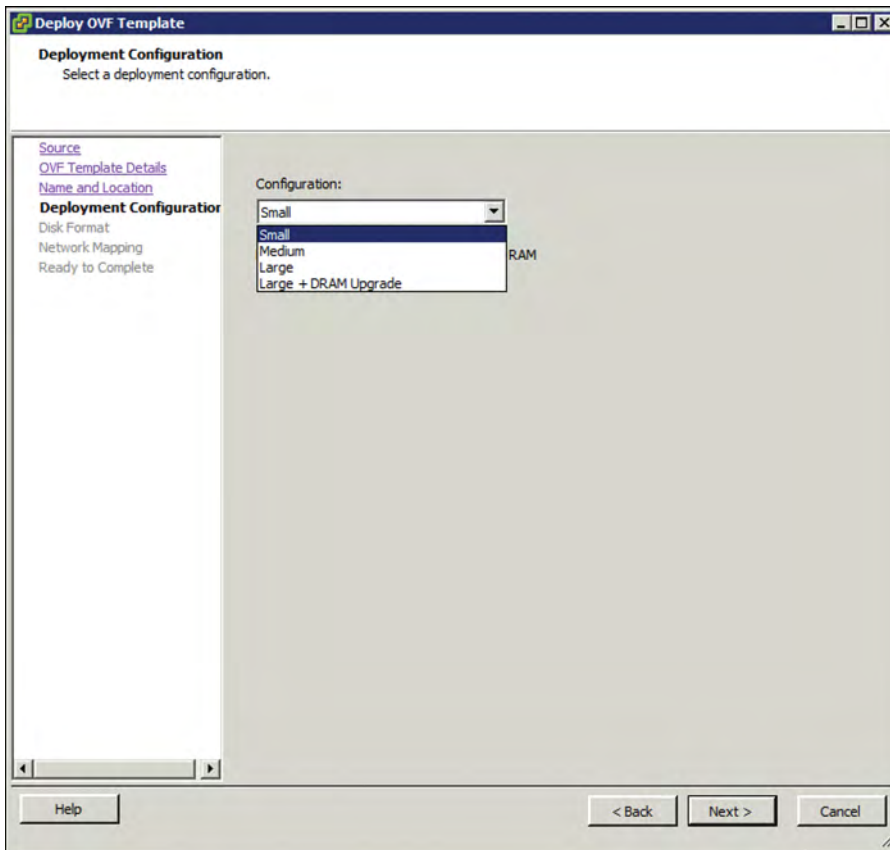


Figure 4-12 *Deploying the OVF Template: Selecting the System Memory Profile for CSR 1000V*

2. Select the appropriate type of disk formatting (see Figure 4-13), and then click Next:
 - **Thick Provision Lazy Zeroed**—With this option, a virtual disk is created with the amount of disk space it has asked for. However, the disk is not cleaned during virtual disk creation. It is cleaned only when you create the first VM on it.
 - **Thick Provision Eager Zeroed**—With this option, a virtual disk is created with the amount of disk space it has asked for. However, the disk is cleaned during virtual disk creation.
 - **Thin Provision**—Choose this option to save space. Initially, the space allocated to a thin disk is less. However, the virtual disk keeps growing as memory requirements grow.

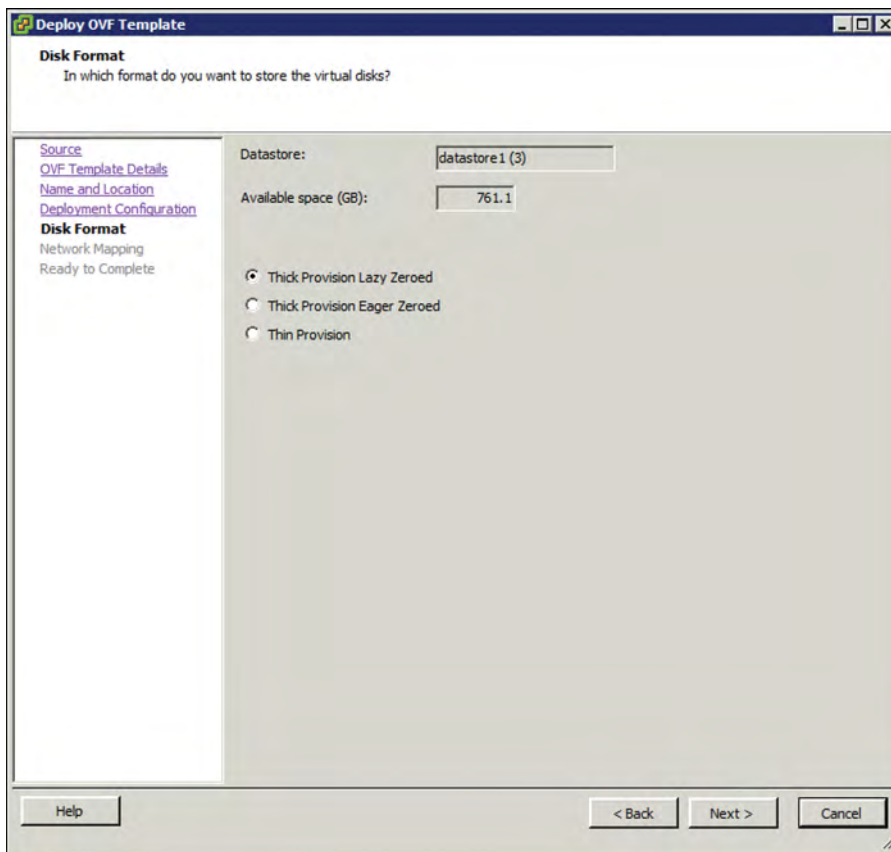


Figure 4-13 *Deploying the OVF Template: Choosing the Disk Provisioning Format*

Note The OVF used here is for version 3.13. You might see variations in the default settings with later versions. Please refer to Cisco release documentation.

3. On the screen shown in Figure 4-14, specify network mapping of the source networks (GigabitEthernet) to the destination networks (VM Network by default) mapping allocation.

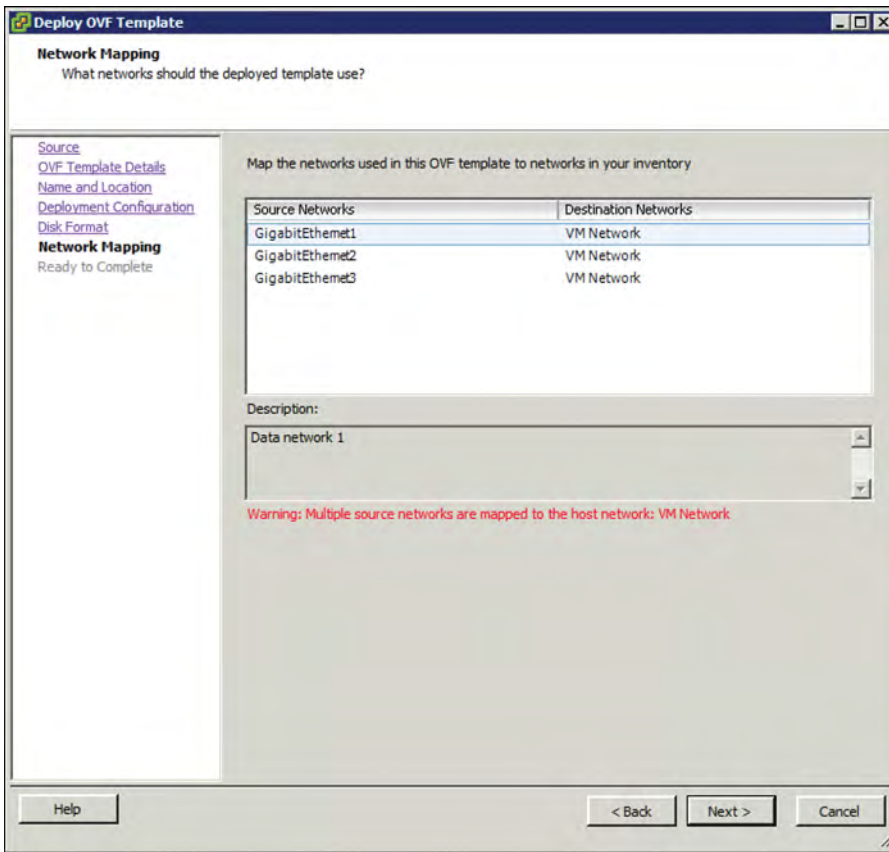


Figure 4-14 Deploying the OVF Template: Network Mapping

4. Look over the summary of the deployed CSR 1000V configuration, as shown in Figure 4-15, and click Finish.

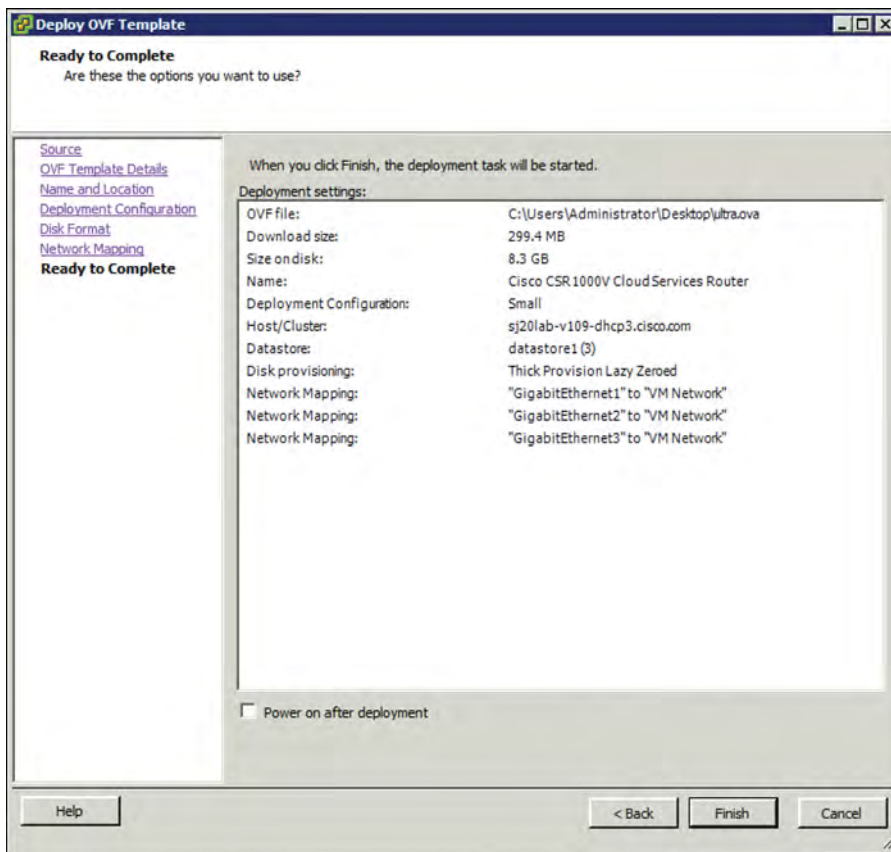


Figure 4-15 Deploying the OVF Template: Checking the Settings

- Step 4.** When the deployment of the CSR 1000V is complete, boot the router by selecting the VGA console from the GRUB menu on the Console tab shown in Figure 4-16.

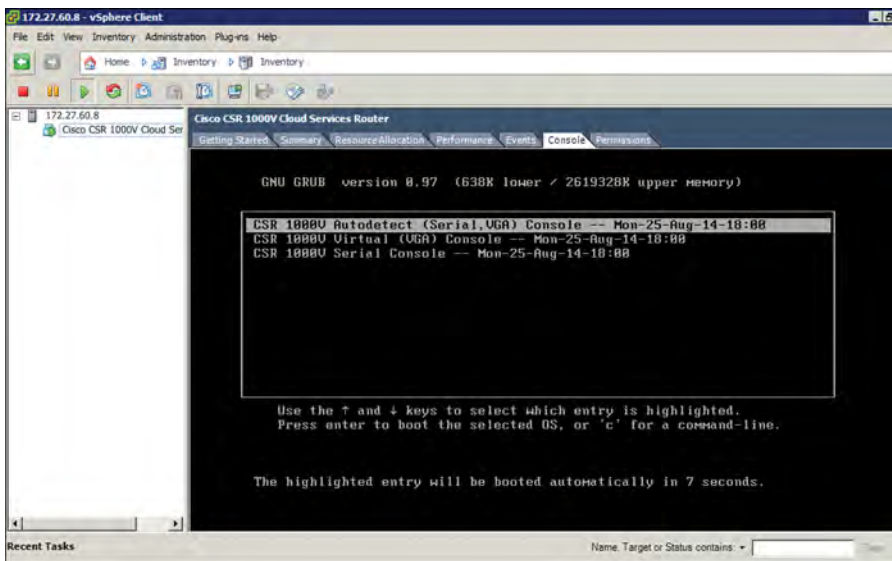


Figure 4-16 CSR 1000V Console Tab

- Step 5.** At the router prompt, enter `platform console serial`, as shown in Figure 4-17. (This command causes the VM to send console information on the serial port from ESXi in the later steps.)

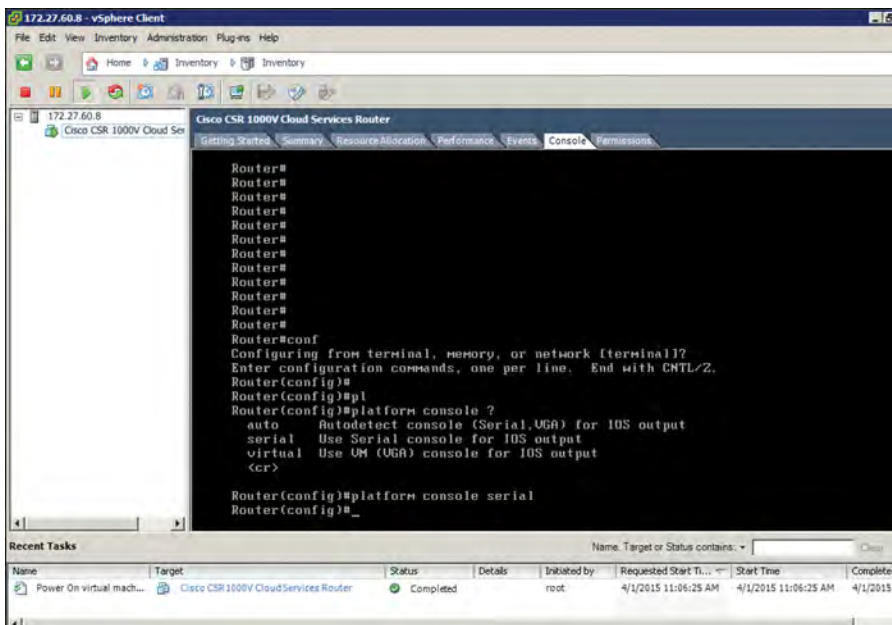


Figure 4-17 CSR 1000V Command Prompt

Step 6. To add the serial port for console access, access the vCenter web client and select Virtual Hardware, Network Adaptor, Serial Port, as shown in Figure 4-18.

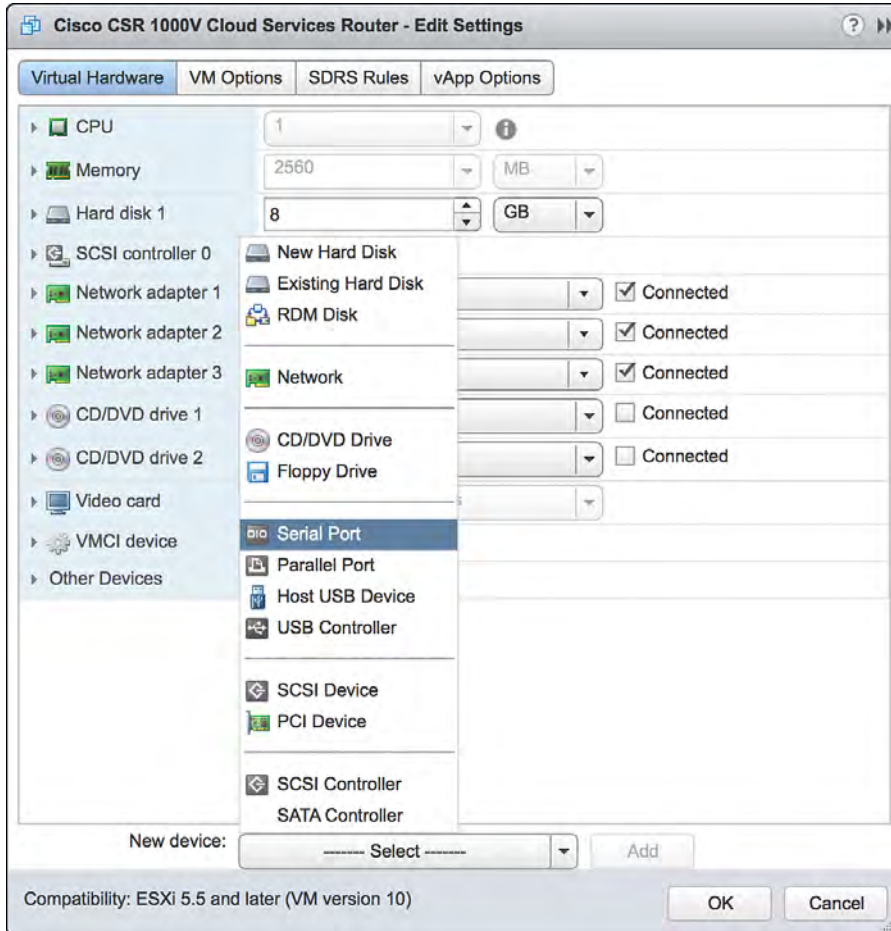


Figure 4-18 VM Access from the vCenter Web Client

Step 7. Shut down the guest OS as shown in Figure 4-19. (Note that this serial port will be used for terminal access to the CSR.)

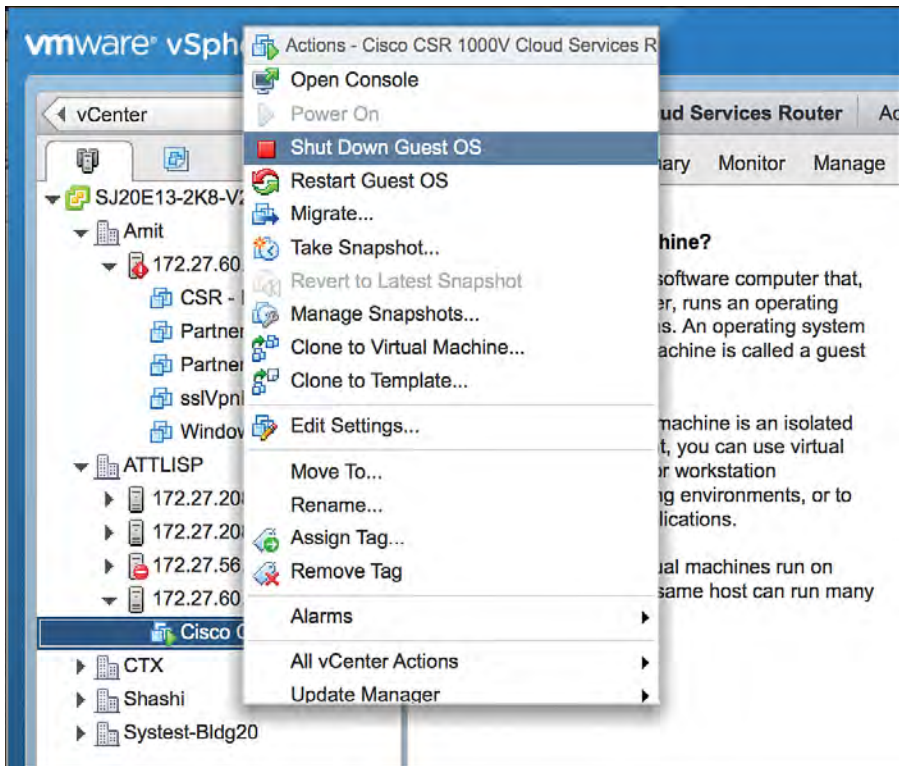


Figure 4-19 *Configuring the Serial Interface: Shutting Down the Router*

- Step 8.** Select Add New Device, New Serial Port and provide the IP address and terminal port details to access the CSR, as shown in Figure 4-20.

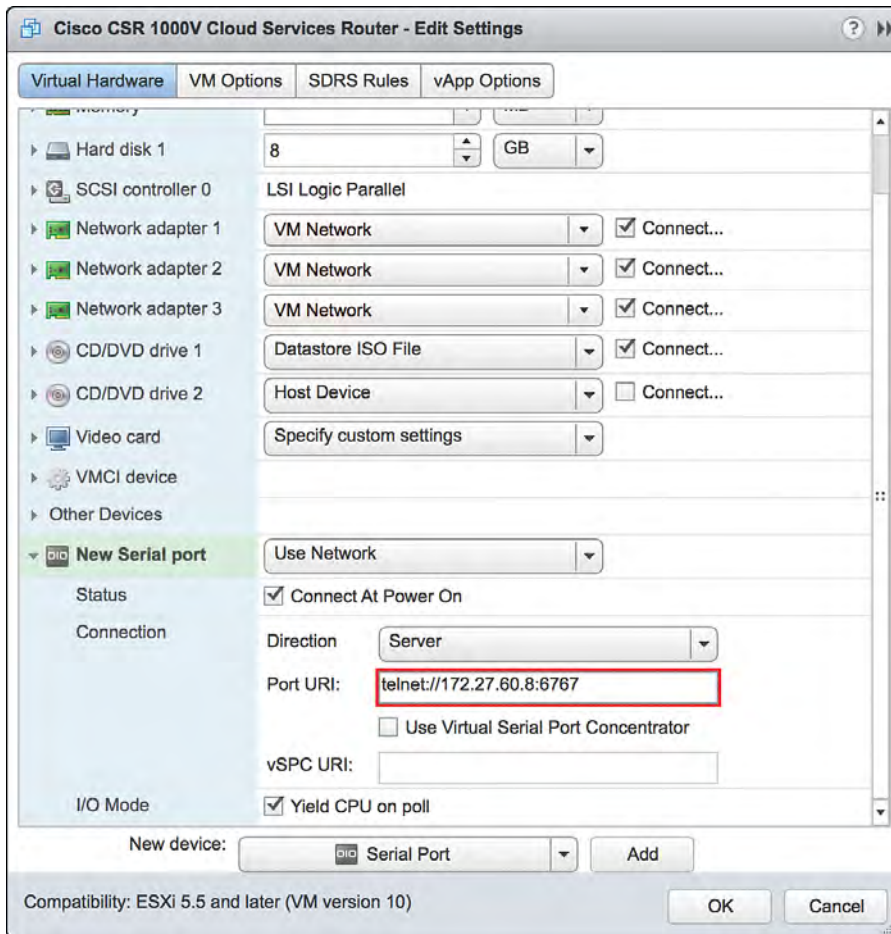


Figure 4-20 *Configuring the Serial Interface: Setting the Telnet Address*

- Step 9.** Go to vCenter and select Setting, Security Profile. Edit security configuration ports 23 and 1024 as shown in Figure 4-21. This is needed because by default ESXi blocks console access.

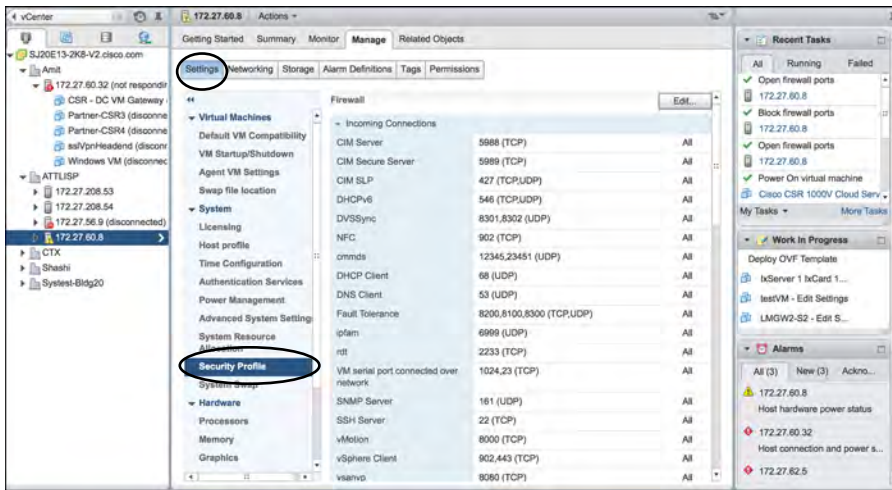


Figure 4-21 *Configuring the Serial Interface: Firewall Settings*

Step 10. Enable ports 23 and 1024 as shown in Figure 4-22.

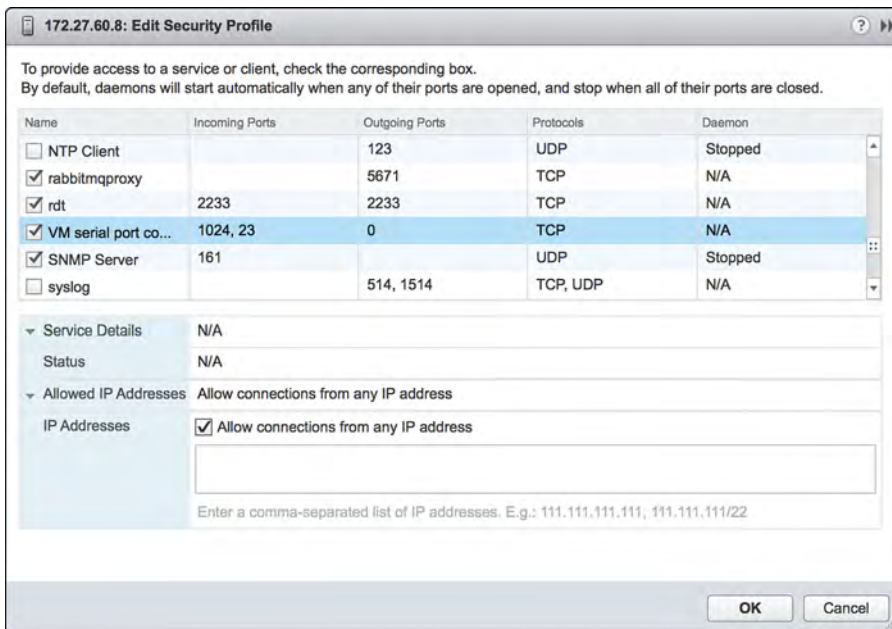


Figure 4-22 *Configuring the Serial Interface: Security Profile Detail*

Step 11. Use Telnet to verify the access from the PC. (It's a good practice to use SSH for accessing the CSR VM; however, for the sake of simplicity, this example shows Telnet access setup.) The EXSi hypervisor defaults the network connections to the VM Network virtual switch connection. The network

adapters are mapped to CSR interfaces. For example, GigabitEthernet1 is mapped to Network adapter 1, and so on. You can verify this by comparing the MAC address as illustrated in Figure 4-23.

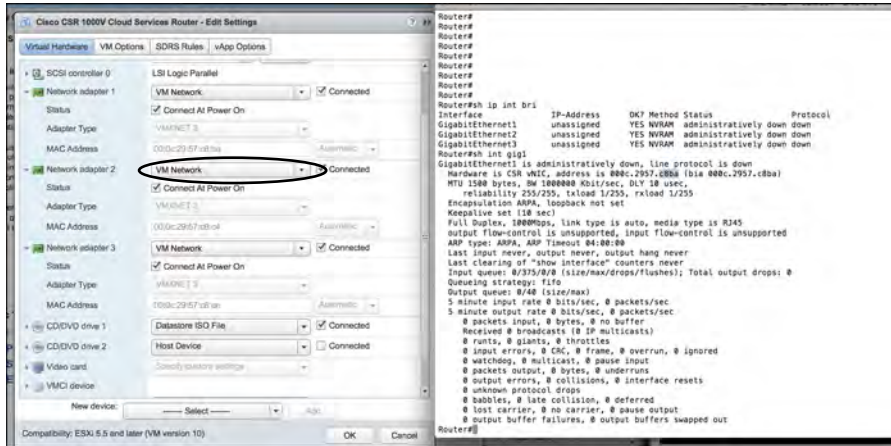


Figure 4-23 CSR 1000V Telnet Access Screen

- Step 12.** To remap the network adapters to corresponding vNICs, you should perform the following steps. From the vSphere client in the Edit Settings window, select New Device Add, Networking and add vNICs to the CSR as assigned interfaces (from the vCenter web client), as shown in Figures 4-24 through 4-27. (Allow all VLANs and create a bridgeForVNIC1 label for this connection.)

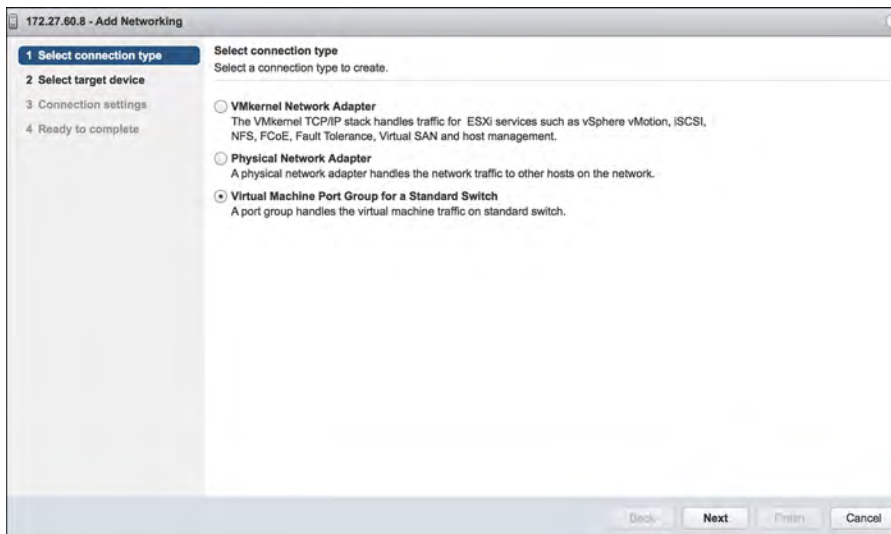


Figure 4-24 vNICs and the CSR 1000V: Selecting the Connection Type

1. Select the new vNIC, as shown in Figure 4-25, to create a new standard switch name.

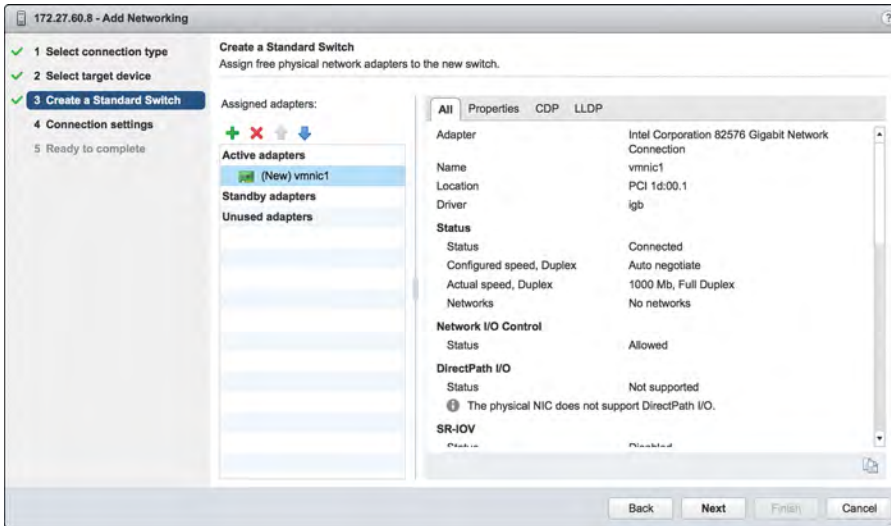


Figure 4-25 *vNICs and the CSR 1000V: Creating a Standard Switch*

2. Add VLANs and the network label assigned for the vNIC, as shown in Figure 4-26.

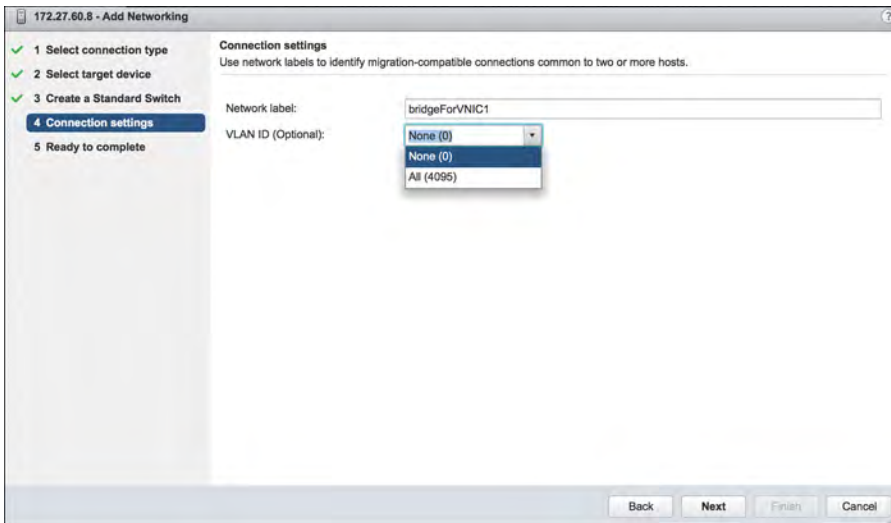


Figure 4-26 *vNICs and the CSR 1000V: Setting the Connection Settings*

3. Complete the configuration of the vNIC with a VLAN and label attachment that can be referenced in a vSwitch. Click Finish to complete this step, as shown in Figure 4-27.

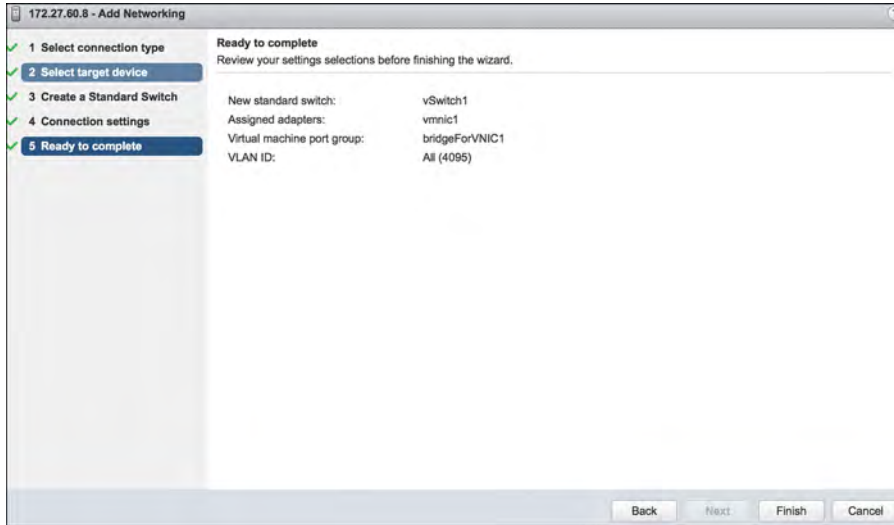


Figure 4-27 *vNICs and the CSR 1000V: Completing the Configuration*

- Step 13.** Go to the vSphere web client and select Virtual Machine, Network Adapter. In the Networking tab, look for the new bridgeForVNIC1 label you created earlier, as shown in Figure 4-28. You should note that this label acts as mapping between the CSR interface and the vNIC.

Repeat Steps 12 and 13 to remap additional network adapters to vNICs available to the CSR.

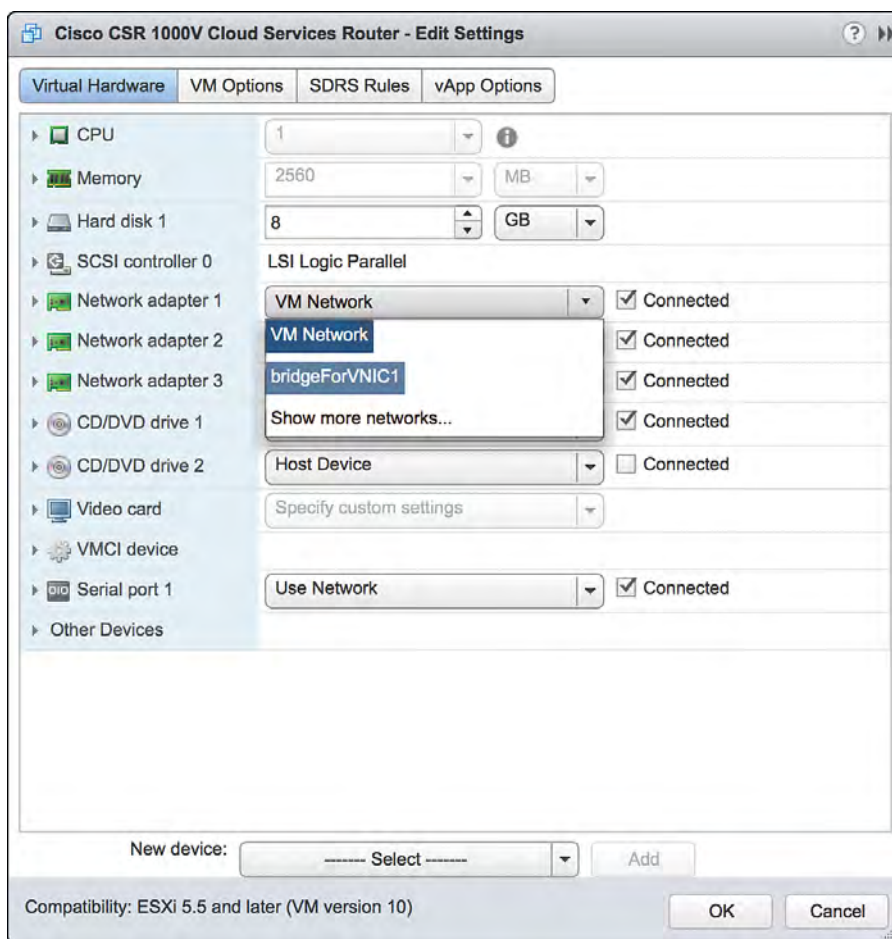


Figure 4-28 *vNICs and the CSR 1000V: Editing the Settings*

To map the network adapter to the vNIC created, select the vNIC label created in the previous step. The CSR 1000V is now configured and connected to the physical NIC, as shown in Figure 4-29.

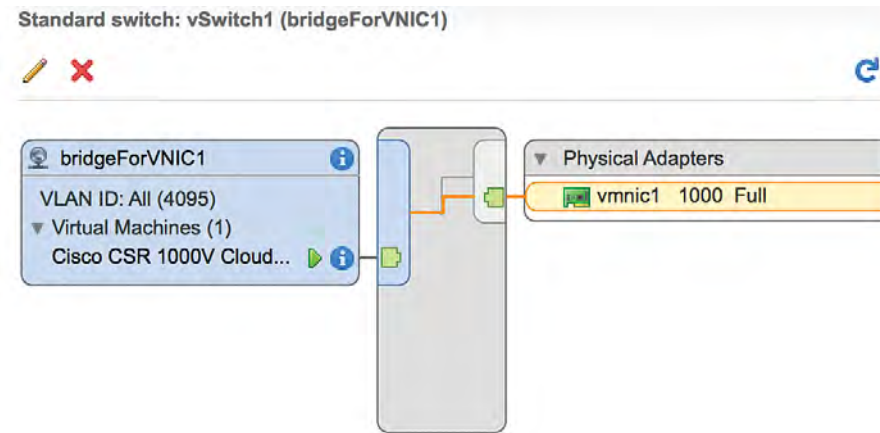


Figure 4-29 vNICs and the CSR 1000V: Interface Summary Screen

Installing the CSR 1000V on a KVM Hypervisor

The process for installing the CSR 1000V on a KVM hypervisor has two phases:

1. Bring up the VM with the CSR 1000V on ESXi.
2. Connect the vNIC with the CSR 1000V.

Bring Up the CSR 1000V as a Guest

Follow these steps to update essential packages on a Linux managed server so it can work as a type 1 hypervisor and run a CSR 1000V VM:

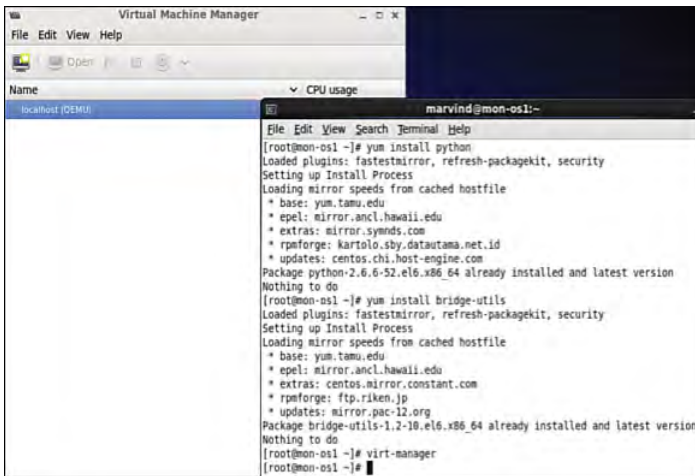
- Step 1.** Install the VM packages `virt-manager`, `qemu-kvm`, and `bridge-utils` like this:

```
apt-get install virt-manager
apt-get install qemu-kvm
apt-get install bridge-utils
```

or like this:

```
yum install virt-manager
yum install qemu-kvm
yum install bridge-utils
```

Figure 4-30 shows the installation of packages required for CSR creation.



```

Virtual Machine Manager
File Edit View Help
Name CPU usage
localhost [DEMU] marvind@mon-os1:~
marvind@mon-os1:~
[root@mon-os1 ~]# yum install python
Loaded plugins: fastestmirror, refresh-packagekit, security
Setting up Install Process
Loading mirror speeds from cached hostfile
* base: yum.tamu.edu
* epel: mirror.acl.hawaii.edu
* extras: mirror.symds.com
* rpmforge: kartolo.sby.datutama.net.id
* updates: centos.chi.host-engine.com
Package python-2.6.6-52.el6.x86_64 already installed and latest version
Nothing to do
[root@mon-os1 ~]# yum install bridge-utils
Loaded plugins: fastestmirror, refresh-packagekit, security
Setting up Install Process
Loading mirror speeds from cached hostfile
* base: yum.tamu.edu
* epel: mirror.acl.hawaii.edu
* extras: centos.mirror.constant.com
* rpmforge: ftp.riken.jp
* updates: mirror.pac-12.org
Package bridge-utils-1.2-10.el6.x86_64 already installed and latest version
Nothing to do
[root@mon-os1 ~]# virt-manager
[root@mon-os1 ~]#

```

Figure 4-30 Package Installation on a KVM Hypervisor

- Step 2.** Launch Virtual Machine Manager, which is the front end to KVM/QEMU that allows installation and management of CSR VMs, by selecting Application, System, Virtual Machine Manager.

Note Virtual Machine Manager could also be on a different path for your Linux server. Figure 4-31 shows the launch of the virtual machine from QEMU. Make sure you have XDesktop installed. Also note that VMM is not a mandatory requirement for using KVM/QEMU, especially when a graphical user interface is not present on a desktop.

Click the Create a New Virtual Machine icon, and the dialog shown in Figure 4-31 appears. Click the Forward button.

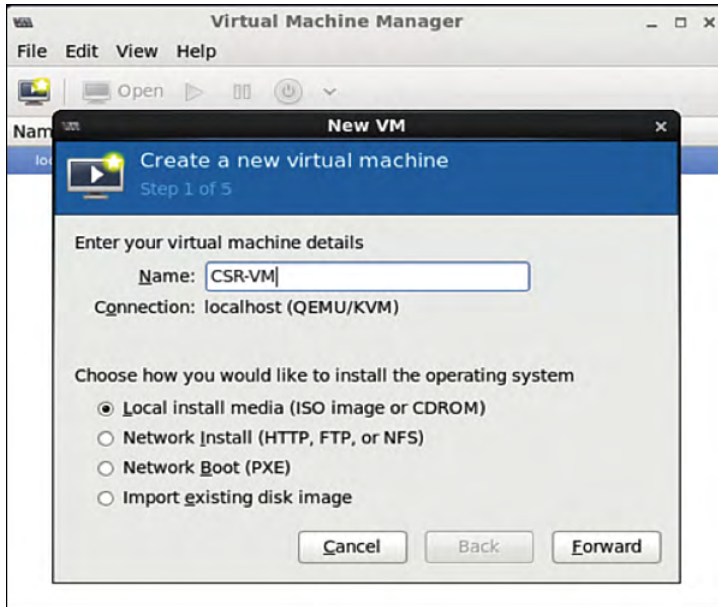


Figure 4-31 *Creating a Guest VM*

- Step 3.** Load the ISO image (which you download from software.cisco.com) for the CSR 1000V, as shown in Figure 4-32. Click the Forward button.

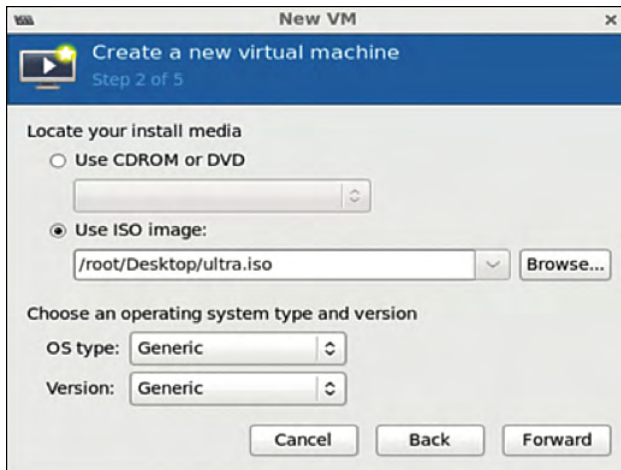


Figure 4-32 *ISO Image Bootup for the CSR 1000V*

Note Download the ISO CSR 1000V image to your local hard disk. When you download it, it is named `csr1000v-universalk9.<version>.std.iso`, but the file is renamed `ultra.iso` in the example shown.

- Step 4.** Allocate hardware resources for the guest VM as shown in Figure 4-33. (Refer to Table 2-2 in Chapter 2 for further allocation information.) Click Forward.

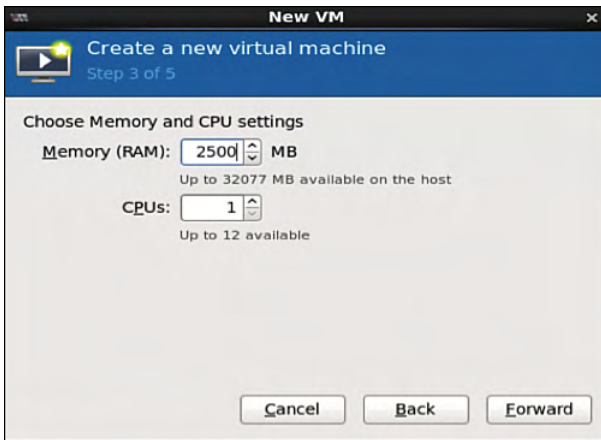


Figure 4-33 *Choosing Memory and CPU Settings*

- Step 5.** Select hardware resources, as shown in Figure 4-34, and click Forward.

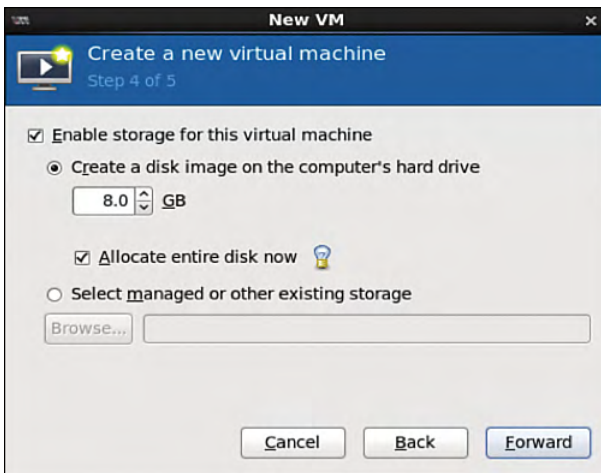


Figure 4-34 *Selecting Hardware Resources*

Note If you do not check Allocate Entire Disk Now, only a small portion of memory asked for will be allocated. It will keep growing as memory requirements increase. Checking Allocate Entire Disk Now guarantees that much storage.

Step 6. Look over the hardware resources summary (see Figure 4-35) and make any changes needed. Click Finish.

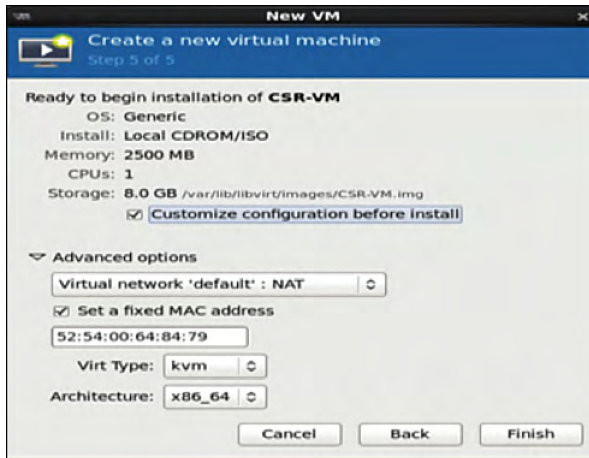


Figure 4-35 Resources Summary Snapshot

Step 7. To apply changes for the guest VM, select Application, System, Virtual Machine Manager and highlight the CSR installed in the VMM. Then click the Show Virtual Hardware Details tab and click the Add Hardware button, as shown in Figure 4-36.



Figure 4-36 *Applying Hardware VM Changes*

- Step 8.** To create serial connection access for console access, select Serial, and then select TCP for Device Type and provide the telnet information, as shown in Figure 4-37.



Figure 4-37 *Creating the Serial Interface*

- Step 9.** In the Virtual Machine Manager, highlight the guest VM and shut it down (if it is not down already). (See Figure 4-38.)

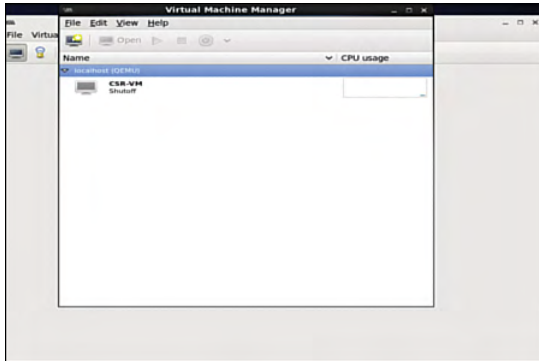


Figure 4-38 *Shutting Down the Guest VM*

The guest VM goes down, as shown in Figure 4-39.

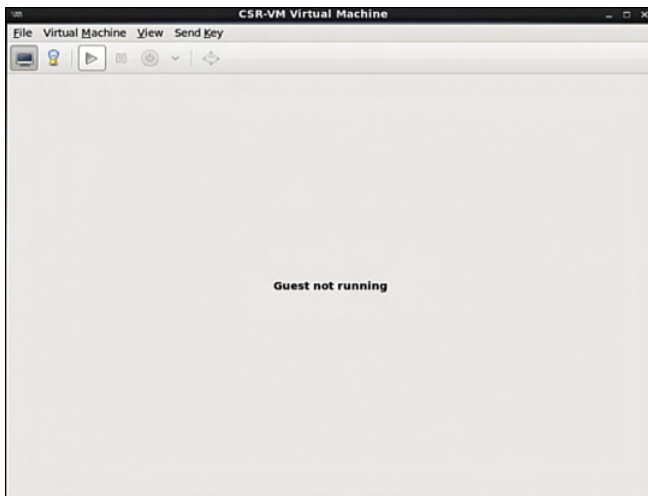


Figure 4-39 *Shutdown of the Guest VM*

- Step 10.** Access the router from the console, as shown in Figure 4-40. Make sure the VM is powered up before you try to access it.


```

CSR-VM Virtual Machine
File Virtual Machine View Send Key

=Apr 21 00:35:31.736: %IOSXE_DIR-6-INSCARD: Card (fp) inserted in slot F0
=Apr 21 00:35:31.736: %IOSXE_DIR-6-ONLINECARD: Card (fp) online in slot F0
=Apr 21 00:35:32.238: %SYS-5-RESTART: System restarted --
Cisco IOS Software, CSR1000U Software (X86_64_LINUX_IOSD-UNIVERSALK9-M), Experimental Version 15.5(20150318:152821) [mcp_dev-BLD-BLD_MCF_DEV_LATEST_20150318_123025-ios 1511]
Copyright (c) 1986-2015 by Cisco Systems, Inc.
Compiled Wed 18-Mar-15 14:19 by mcpre
=Apr 21 00:35:32.381: %CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is OFF
=Apr 21 00:35:32.382: %CRYPTO-6-GDOI_ON_OFF: GDOI is OFF
Router>
=Apr 21 00:35:45.049: %UPAN-5-PACKAGE_SIGNING_LEVEL_ON_INSTALL: F0: vman: Package 'iosxe-remote-mgmt.BLD_MCF_DEV_LATEST_20150318_123025.ova' for service container 'csr_mgmt' is 'Cisco signed', signing level cached on original install is 'Cisco signed'
Router>
Router>
=Apr 21 00:35:46.295: %VIRT_SERVICE-5-INSTALL_STATE: Successfully installed virtual service csr_mgmt
Building configuration...
=Apr 21 00:35:46.566: %ONEP_BASE-6-SS_ENABLED: ONEP: Service set Base was enabled by Default[OK]
=Apr 21 00:35:49.972: %CONFIG_CSRLXC-5-CONFIG_DONE: Configuration was applied and saved to NVRAM. See bootflash:/csr1xc-cfg.log for more details._

```

Figure 4-40 Console Access to the KVM

Step 11. Use the serial interface command for telnet access: `platform console serial` and write `mem`, as shown in Figure 4-41.

```

CSR-VM Virtual Machine
File Virtual Machine View Send Key

Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#conf
Configuring from terminal, memory, or network [terminal]?
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#ip conb
Router(config)#pla
Router(config)#platform c
Router(config)#platform con
Router(config)#platform console serial
Router(config)#
Router#
=Apr 21 00:36:36.054: %SYS-5-CONFIG_I: Configured from console by console_

```

Figure 4-41 Router Console for Telnet Access

Step 12. Access the CSR 1000V via the telnet, as shown in Figure 4-42.



```

Terminal
File Edit View Search Terminal Help
to administratively d
Building configuration...
bnw
*Apr 21 00:43:30.239: %SYS-6-BOOTTIME: Time taken to reboot after reload = 385
seconds
*Apr 21 00:43:33.174: %VMAN-5-PACKAGE_SIGNING_LEVEL_ON_INSTALL: F0: vman: Packa
ge 'iosxe-remote-mgmt.BLD_MCP_DEV_LATEST_20150318_123025.ova' for service contai
ner 'csr_mgmt' is 'Cisco signed', matches signing level cached on original insta
ll, signing level allowed is 'Cisco signed'
*Apr 21 00:43:33.313: %VIRT_SERVICE-5-INSTALL_STATE: Successfully installed virt
ual service csr_mgmt
*Apr 21 00:43:35.059: %ONEP_BASE-6-SS_ENABLED: ONEP: Service set Base was enable
d by Default[OK]
*Apr 21 00:43:40.671: %CONFIG_CSRLXC-5-CONFIG_DONE: Configuration was applied an
d saved to NVRAM. See bootflash:/csrlxc-cfg.log for more details.%IOSXEBOOT-4-NE
T-SMP-AFFINITY: (local/local): reached 2 minutes limit for broken_parity_status
IOSXEBOOT-4-NET-SMP-AFFINITY: (rp/0): reached 2 minutes limit for broken_parity_
status
Router>
Router>
Router>
Router>

```

Figure 4-42 *Telnet Connection to the CSR 1000V*

- Step 13.** Ensure that your virtual machine is shut down, and then start vNIC provisioning by selecting Show Virtual Hardware Details, NIC, as shown in Figure 4-43.

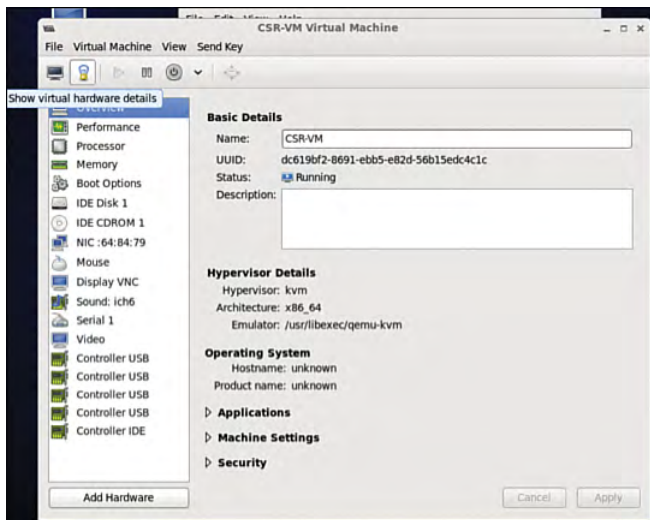


Figure 4-43 *Accessing CSR 1000V Network Settings*

- Step 14.** In the Virtual Machine Manager, select virtio as the device model (see Figure 4-44) because it is the para-virtualized driver in Linux. Using virtio is the best way to exploit the underlying kernel for I/O virtualization. It provides an efficient abstraction for hypervisors and a common set of I/O drivers.

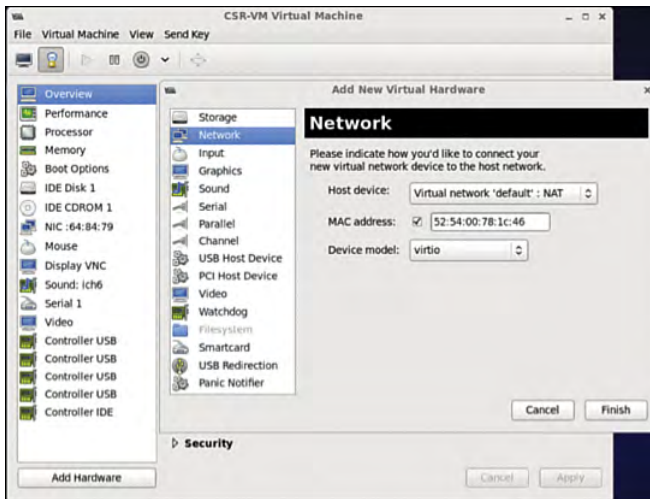


Figure 4-44 *Selecting CSR 1000V Network Settings*

Select the virtual network with NAT to tie all VMs in the same bridge domain and NAT it to the outgoing physical interface (see Figure 4-45). Attach the other NIC to the bridge tap.

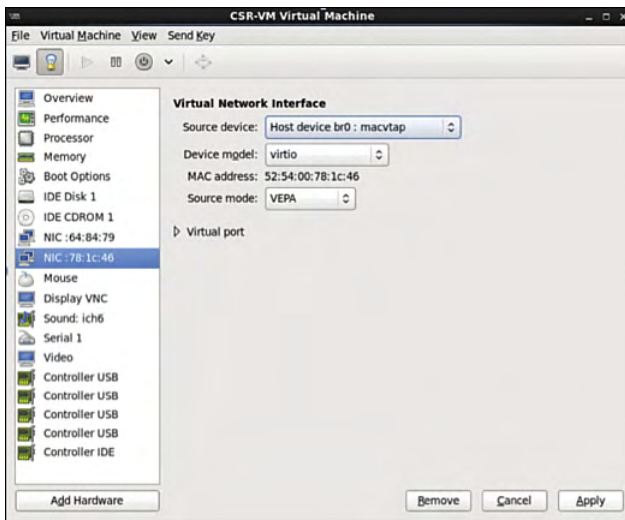


Figure 4-45 *CSR 1000V NIC Settings*

In KVM, `macvtap` is a combination of the `macvlan` driver and a Tap device. Here the function of the `macvlan` driver is to create virtual interfaces and map virtual interfaces to physical network interfaces. A unique MAC address identifies each virtual interface to the physical interface. A TAP interface is a software only interface that exists only in the kernel. You use Tap interfaces to

enable user-space networking and allow passing of datagrams directly between VMs instead of sending datagrams to and from a physical interface. The `macvtap` interface combines these two functions together (see Figure 4-46).

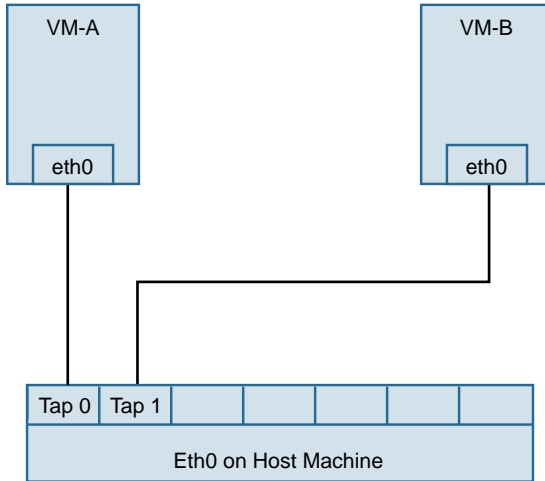


Figure 4-46 `macvtap` Diagram

Step 15. Configure the mapping of the vNIC to the physical interface:

1. Access the directory `/etc/network/interfaces/ifcfg-br0` on the Ubuntu host and view the bridge type (see Figure 4-47).

```

Terminal
File Edit View Search Terminal Help
DEVICE=br0
TYPE=Bridge
BOOTPROTO=static
ONBOOT=yes
TYPE=bridge
STP=on

*ifcfg-br0* 6L, 70C

```

Figure 4-47 Bridge Configuration File Output

2. Access the directory `/etc/network/interfaces/ifcfg-eth4` and configure the vNIC to be in the same bridge type, BR0 (see Figure 4-48).

```

terminal
File Edit View Search Terminal Help
DEVICE=eth4
HWADDR=78:81:05:FF:A5:0E
TYPE=Ethernet
UUID=48551308-8180-4637-9d17-75378a141f96
ONBOOT=no
NM_CONTROLLED=yes
BOOTPROTO=dhcp
IPADDR=9.1.1.1
NETMASK=255.255.255.0
BRIDGE=br0
*ifcfg-eth4* 10L, 184C

```

Figure 4-48 *Interface Configuration File Output*

To configure the spanning tree mode to promiscuous, use this:

```

auto eth4
iface eth4 inet manual
up ip address add 0/0 dev $IFACE
up ip link set $IFACE up
up ip link set $IFACE promisc on

```

Alternatively, access the file `/etc/network/interfaces/ifcfg-eth4` and type this:

```
PROMISC=yes
```

This method provides persistent configuration settings for `ifcfg-eth4`.

- Step 16.** In the Virtual Machine Manager, select Show Virtual Hardware Details.

Performance Tuning of the CSR 1000V

To improve performance of a guest VM in a hypervisor environment, you improve availability of the I/O and other hardware resources through para-virtualization. Para-virtualization allows for a kernel to present a software interface to a guest VM that is similar but not identical to that of the underlying hardware, thereby improving the VM performance. If you want to tune the performance further, you need to look at two components:

- Hypervisor scheduling
- CPU pinning

This section reviews the common tuning practices for an ESXi host. The scheduler for ESXi is responsible for vCPU, IRQ (interrupt requests), and I/O threads. To provide equal treatment to all guest VMs, the scheduler provides allocation of equal resources of vCPU threads for scheduling. Note that you can relax coscheduling of threads to avoid synchronization latency.

To tweak the scheduling and resource allocation details, you must access the VM setting using vSphere client and follow these steps:

1. In the vSphere client inventory, right-click the virtual machine and select Edit Settings.
2. Click the Resources tab and select CPU.
3. Allocate the CPU capacity for this virtual machine.

The Processor Affinity setting (CPU pinning) restricts VMs to a particular set of cores by defining the affinity set. The scheduling algorithm aligns with process affinity for assigning the resources used for the tasks. Figure 4-49 assumes two tasks: Task 1 and Task 2. Task 1 has affinity to processor 1 and is using it. When Task 2 needs a resource, the scheduler uses a second processor. Task 2 then acquires affinity with the second processor.

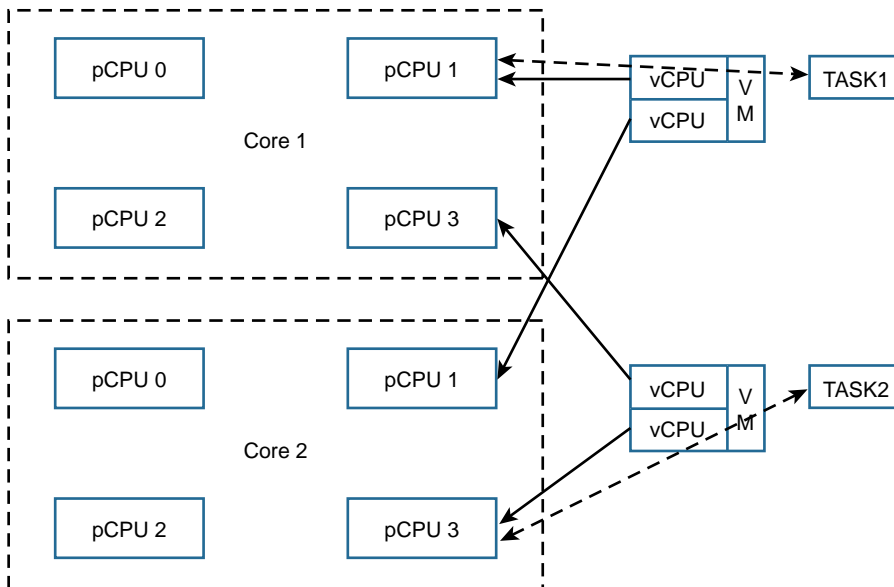


Figure 4-49 CPU Pining

To tweak these settings, access the vSphere client and follow these steps:

1. In the vSphere client inventory panel, select a virtual machine and select Edit Settings.
2. Select the Resources tab and select Advanced CPU.
3. Click the Run on Processor(s) button.

You achieve CPU pinning in KVM by issuing the following command:

```
sudo virsh vcpupin test 0 6
```

Hyperthreading by definition allows a single physical core to have two logical cores; that is, a single core can execute two threads at a given time. Each process from the guest VM can be split into multiple threads to a logical CPU, and the CPU can handle multiple threads of independent tasks. The main function of hyperthreading is to increase the number of tasks in the pipeline by creating parallel pipelines. By tweaking the process affinity option, you can restrict VMs to a particular set of cores and unhook the VM from processor scheduling. Most of the hypervisors use BIOS settings to modify the hyperthreading feature.

For predictable performance, the following best practices are recommended:

- Ensure that hyperthreading is turned off.
- Use CPU pinning to allow the guest VMs to dedicate one or more physical hardware CPUs for processing.
- For CSR 1000V performance optimization, it is important to understand the concept of DirectPath I/O and SR-IOV (single root I/O virtualization). These are driver virtualization and are beneficial for achieving very high packet rates with low latency. In DirectPath I/O, you can map only one physical function to one virtual machine. SR-IOV allows an admin to share a single physical device, so that multiple virtual machines can connect directly to the physical function.

These features are supported in all hypervisors, and it is important to understand the settings on the hypervisor deployed in order to optimize guest VM performance with features used on the hypervisor.

Summary

Now that you've read this chapter, you should have an understanding of the CSR 1000V data plane architecture, as well as packet flow. You should also have an understanding of the steps for bringing up a CSR 1000V on ESXi and KVM hypervisors.

This page intentionally left blank

CSR 1000V Deployment Scenarios

This chapter introduces some of the common deployment scenarios for the CSR 1000V software router. After reading this chapter, you will be able to apply the CSR 1000V to multiple deployment scenarios, such as VPN services extension, route reflector design, branch designs, and Locator/ID Separation Protocol (LISP).

VPN Services

A virtual private network (VPN) is an extension of a private network that enables an organization to securely deliver data services across a public or untrusted network infrastructure such as the Internet. A VPN connection is a logical connection and can be made at either Layer 2 or Layer 3 of the OSI (Open System Interconnect) model. In today's environment, a VPN typically uses encryption for data privacy and integrity protection.

Layer 2 VPNs

Layer 2 VPNs (L2VPN), as the name suggests, operate at Layer 2 of the OSI model. They provide virtual circuit connections that are either point-to-point or point-to-multipoint. The forwarding of traffic in L2VPN is based on Layer 2 header information such as MAC address. One of the advantages of L2VPN is that it is agnostic to the Layer 3 traffic protocol that is carried over it because the L2VPN operates at a lower OSI layer.

The following are some examples of L2VPN services:

- **Point-to-point L2VPNs**—Frame Relay, ATM PVC, Layer 2 Tunneling Protocol version 3 (L2TPv3), and Virtual Private Wire Service (VPWS)
- **Multipoint L2VPNs**—Virtual Private LAN Service (VPLS) and Provider Backbone Bridging Ethernet Virtual Private Network (PBB-EVPN)

Layer 3 VPNs

Layer 3 VPNs (L3VPN) provide IP connectivity between sites and operate at Layer 3 of the OSI model. L3VPNs have many flavors and can be either point-to-point or multi-point connections for site-to-site route exchanges.

The following are some examples of L3VPN technologies:

- MPLS VPNs**—Multiprotocol Label Switching (MPLS) offers a cost-effective method for replacing dedicated leased-line circuits such as Frame Relay or ATM networks. L3 MPLS VPN service requires that the enterprise peer with the service provider (SP) at the IP Layer 3 level. In this scenario, the SP network is involved in the routing of the IP packets from the enterprise. This capability, coupled with Virtual Routing and Forwarding (VRF), allows the SP to provide traffic segmentation between the customer's IT services. VRF is a technology embedded in routers that allows multiple instances of a routing table, one for each VPN, to exist on a provider edge (PE) router. Because the routing instances are independent, the same or overlapping IP addresses can be used without conflict. MPLS L3VPNs offer several advantages, including flexibility, scalability, QoS, and any-to-any connectivity. Figure 5-1 shows different components of an MPLS L3VPN network.

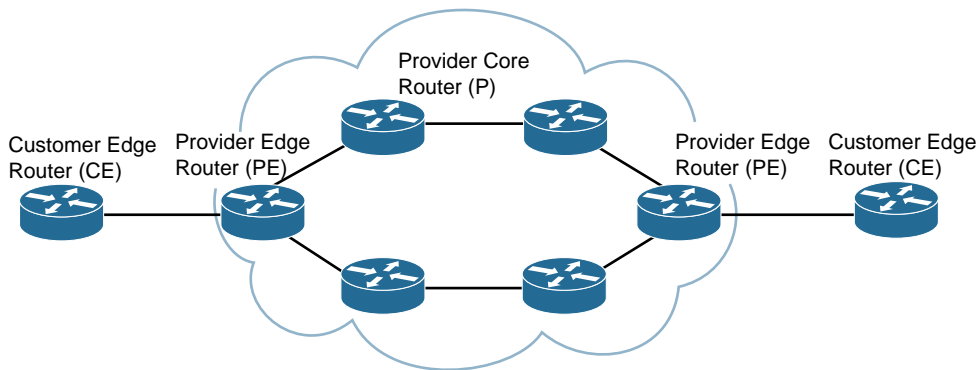


Figure 5-1 *MPLS L3VPN Components*

- GRE tunnels**—Generic Routing Encapsulation (GRE) is a standardized tunneling protocol originally developed by Cisco for encapsulating non-IP protocols (such as IPX and AppleTalk) and transporting them across an IP network. GRE tunnels provide virtual point-to-point links between sites, allowing the sites to see one another's private networks as if they were all one homogenous connection. GRE tunnels are capable of transporting multiple protocols and no routable protocols across an IP network. For example, MPLS frames are not routable over an IP network but can be encapsulated inside a GRE tunnel for transport across an IP core.
- IPsec VPNs**—IP Security (IPsec) is a suite of protocols that provides data privacy and integrity protection. IPsec uses cryptographic security services to ensure that communications over public networks such as the Internet stay confidential. As

described in the following sections, IPsec VPNs can be classified into two categories: site-to-site VPNs and remote access VPNs.

Site-to-Site VPNs

Site-to-site VPNs offer secure connections between enterprise branch locations and enable secure communication between the branch offices, the head office, and data center. Site-to-site VPNs offer several advantages:

- They provide higher-bandwidth performance by using dedicated network devices for hardware encryption.
- The IP addresses of branch LANs and head office/data center's network information are hidden inside IPsec-encrypted headers, away from external prying eyes.
- They offer greater scalability, because it is easy to add a new site or an office to the network, and it is very cost-effective as well because a new office can leverage public Internet circuits for connection over the Internet.

There are several flavors of site-to-site IPsec VPNs, each with unique characteristics. The following sections cover the various IPsec VPN services supported on the CSR platform.

Classic IPsec with Crypto Maps

A classic IPsec VPN with a static crypto map offers the most basic secure transport option for data traffic. It protects unicast traffic from one subnet to another and is usually the lowest common denominator for interoperability between devices from different vendors. The IPsec encryption policy is applied to the egress interface where the encryption process is the last function to be applied to the data traffic.

The crypto access control list (ACL) in the IPsec policy determines what traffic will be encrypted and what traffic will be left alone, without encryption. The data traffic matching the permit statement in the crypto ACL is the “interesting traffic” to which the encryption algorithm is applied. Network administrators must explicitly define a protection profile for every potential subnet that requires data encryption protection.

Classic IPsec is sometimes referred to as *static routing VPN* or *policy IPsec VPN* because it is statically configured based on the security policy defined by network administrators. The IPsec protection profile in the crypto map essentially acts as routing entries and determines which traffic will be sent over the encrypted tunnel, while the rest of the traffic will follow the existing routing rules. However, there are complexities to be aware of with IPsec profiles. For example, the addition of a single subnet in the VPN network requires configuration updates to the other VPN gateways in the network that interact with the subnet. In addition, classic IPsec with crypto map lacks support for IP multicast traffic as the original IPsec RFCs did not accommodate multicast traffic in the IPsec requirement.

Because classic IPsec requires a network admin to explicitly define every potential IP flow on every VPN gateway and because of the lack of multicast traffic support, this is not a viable method for building a large complex network for an enterprise with hundreds or thousands of subnets. A large IPsec VPN may require N^2 IPsec crypto profile configurations at the head end VPN gateway.

Dynamic Multipoint VPN (DMVPN)

DMVPN is the preferred IPsec VPN solution for enterprises requiring encrypted connectivity between branch offices as it offers dynamic spoke-to-spoke communication without the need to build fully meshed tunnel configurations. DMVPN enables establishment of direct spoke-to-spoke connectivity over IPsec on demand, without having the traffic hairpin at the hub location.

DMVPN offers a scalable IPsec VPN solution with flexible deployment topology. It supports hub-and-spoke, partial-mesh, or dynamic full-mesh topology. DMVPN combines several standards-based protocols:

- IPsec
- Multipoint GRE (mGRE) interface
- Next Hop Resolution Protocol (NHRP)

The DMVPN architecture is based on remote spokes establishing connections into a headend VPN gateway, forming dynamic adjacencies between the hub and the spokes. The hub does not need to know any information about the remote spokes ahead of time as that information is exchanged during the initial spoke-to-hub registration. Through the spoke-to-hub GRE over IPsec tunnel, the remote spokes dynamically advertise the tunnel address and the LAN subnets behind the spoke router. The destination LAN addresses and their next hops are learned through routing protocols such as OSPF, EIGRP, and BGP. Figure 5-2 shows the high-level architecture for DMVPN.

DMVPN uses the mGRE interface, which is a point-to-multipoint interface where a single tunnel interface can terminate multiple GRE tunnels from the spokes. This effectively reduces configuration complexity and allows incoming GRE tunnel connections from any spoke, including the ones using dynamically allocated tunnel addresses. The mGRE interface drastically simplifies the hub setup, allowing the hub tunnel interface configuration to stay the same while more spoke routers are incrementally added.

The hub identifies a remote spoke's tunnel endpoint address through NHRP. When the remote peer builds the permanent tunnel to the hub router, it also sends an NHRP registration message to the hub over the tunnel. The NHRP registration message identifies the tunnel endpoint address of the spoke router, thus preventing the hub router from having to know the remote peer's tunnel IP address in advance. Leveraging NHRP messages provides the freedom for the spoke router to use dynamically assigned addresses as well as to build dynamic site-to-site tunnels between the spokes for traffic communication between the branches. The DMVPN topology is shown in Figure 5-2.

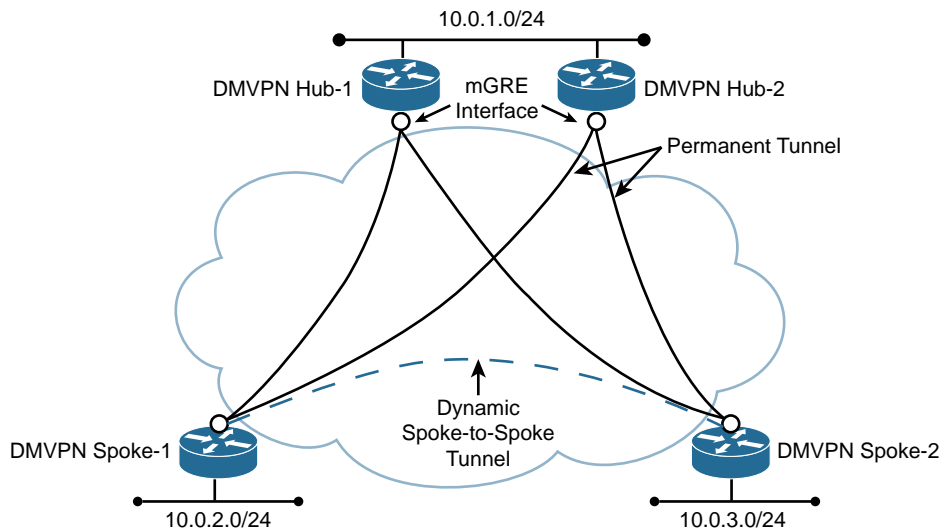


Figure 5-2 *Dynamic Multipoint VPN (DMVPN) Topology*

From a high-availability standpoint, DMVPN offers active-active redundancy. Dual WAN links and redundant hub routers provide a high-availability DMVPN design. DMVPN supports dual-hub design, where the spoke routers are peered with two hubs, providing active-active redundancy where both hubs can be used simultaneously, offering rapid failover if one of the hubs fails.

DMVPN supports hierarchical hub deployment and allows a network to scale to tens of thousands of branch nodes while offering any-to-any communication between the branch sites. Incrementally adding more hubs for additional throughput performance or scaling requirements can gain additional performance.

Group Encrypted Transport VPN (GET VPN)

GET VPN addresses the need for enterprises to provide data encryption over private WANs managed by service providers. Such encryption is often motivated by a regulatory compliance requirement such as the Health Insurance Portability and Accountability Act (HIPAA) and the Payment Card Industry Data Security Standards (PCI DSS).

GET VPN is a tunnel-less VPN technology that offers end-to-end encryption for data traffic and maintains full-mesh IPsec VPN topology without prior negotiation of point-to-point tunnels, as shown in Figure 5-3. Like the other VPN technologies discussed so far in this chapter, GET VPN leverages the same IPsec protocol suites to provide data confidentiality and integrity protection. In addition, it introduces Group Domain of Interpretation (GDOI), an IETF standards-based protocol (RFC 6407), as the security key management protocol.

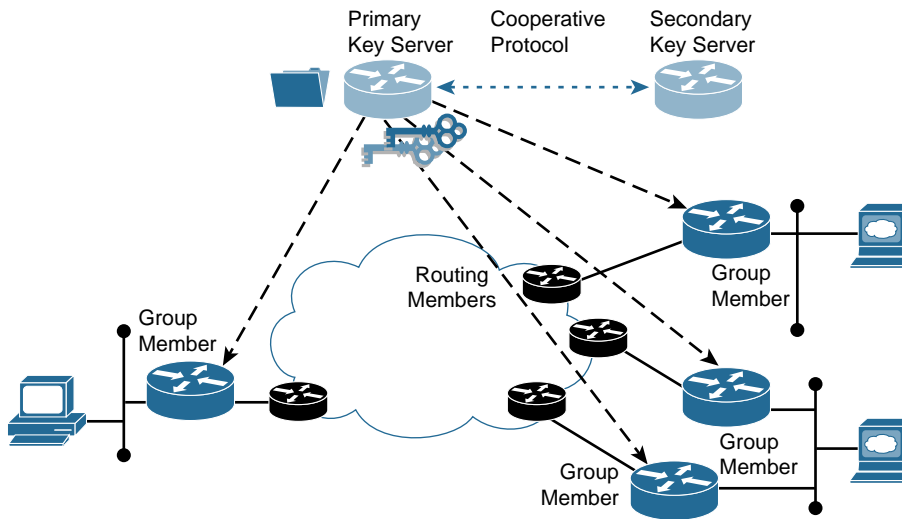


Figure 5-3 GET VPN Components

Two group security functions have been introduced for devices participating in GET VPN communication:

- A *Group Member (GM)* is a device that is responsible for securing data traffic. It is in charge of encrypting and decrypting the data traffic. The GM registers with a Key Server to obtain key materials for data encryption and decryption.
- A *Key Server (KS)* is responsible for authenticating the GMs, managing the security policies, creating group keys (GK), and distributing the GK material to the GMs. On a periodic basis, a KS would send out new key material to refresh the keys prior to their expiration. The most important function for a KS is the generation of the encryption keys. A KS generates two types of keys:
 - **Key-Encryption-Key (KEK)**—KEK is used to secure the control plane messaging between the KS and the GMs.
 - **Traffic Encryption Key (TEK)**—This key is used for encrypting the data traffic between the GMs.

Unlike the traditional IPsec VPN model, which is a bilateral trust model in which a pair of VPN gateways mutually authenticate each other and set up IPsec sessions between them, an important aspect of GET VPN is that it does not set up any IPsec tunnel between the GMs. GET VPN uses a group trust model in which every GM shares the same security policy and encryption keys obtained from the KS. The security policy defines what traffic will be encrypted, the encryption algorithm to use, and the encryption keys to use.

GET VPN uses a tunnel mode called “address preservation,” which copies the original source and destination IP addresses from the original IP header. Address preservation allows packets to traverse asymmetric paths in a private MPLS WAN network. In

In addition to the IP address, other fields from the original IP header, such as the Type of Service (ToS), Identity (Id), and Don't Fragment (DF) fields, are also preserved. The interesting property of address preservation allows IP multicast packets to be routed natively over the provider network because GET VPN applies this method to both multicast and unicast traffic for optimal packet delivery.

High availability is achieved by using a set of KSs running in cooperative mode, where they jointly accept registration from GMs and distribute GDOI rekeys. The Cooperative Key Server Protocol enables the KSs to communicate among themselves and exchange active group policy and encryption keys. If the primary KS becomes unreachable, the remaining KS continues to distribute group policy and group keys to the GMs. This ensures that the GET VPN encryption domain is uninterrupted and continues to function as long as one of the KSs is reachable.

The group trust model for GET VPN is most suitable when the VPN gateways are part of the same network domain and all VPN gateways are trusted to decrypt any packet encrypted and forwarded by other GET VPN gateways. It leverages a centralized entity, the KS, for authentication and distribution of security policy and encryption keys. Most importantly, Network Address Translation (NAT) is not present along the network paths between the GMs or KSs. These characteristics are usually found in MPLS networks. If there are NAT devices present along the network paths between the GMs, then the network is not suitable for deploying GET VPN.

Remote Access VPNs

The CSR 1000V offers a Secure Sockets Layer (SSL) VPN gateway that allows remote user access to corporate data resources and empowers employees to work from anywhere on corporate laptops as well as personal mobile devices, regardless of physical location.

With the CSR SSL VPN solution, in conjunction with the Cisco AnyConnect VPN client, end users gain access securely from home, from the road over 4G and wireless hotspots, or from any Internet-enabled location. SSL VPN offers three modes of SSL VPN access (though only the tunnel mode is supported with a CSR SSL VPN gateway):

- *Clientless mode* provides secure access to web resources only and is useful for accessing resources that are on corporate web servers.
- *Thin client mode* extends the capability of secure web access, with a port-forwarding Java applet allowing remote access to TCP-based applications such as POP3, SMTP, IMAP, and SSH that are not web-based protocols.
- *Tunnel mode* delivers a lightweight and centrally configured SSL VPN service that offers extensive application support through a Cisco AnyConnect Secure Mobility Client. Full tunnel mode provides secure network access to virtually any applications on the enterprise network. During the establishment of the VPN with the CSR SSL VPN gateway, the Cisco AnyConnect Secure Mobility Client is downloaded and installed on the remote user device. When the user logs into the SSL VPN gateway, it establishes the tunnel connection, and the network access is determined by the group policy configured on the gateway. After the user closes

the connection, the AnyConnect Secure Mobility Client is removed from the client device by default; however, there is an option to keep the AnyConnect Secure Mobility Client installed on the client equipment.

Use Cases for the CSR 1000V as a VPN Service Gateway

The following sections describe common use cases for the CSR 1000V functioning as a VPN gateway in an enterprise environment.

Enterprise Data Center Network Extension

One approach to cloud data center access is to provision a single VPN backhaul connection between an existing data center and the data center in the cloud. This type of solution is simple to set up, but the drawback is that all the traffic to the data center in the cloud requires backhauling through the existing data center. This can potentially increase latency and may require an expensive private WAN link between the two data centers.

The CSR 1000V as a VPN Gateway

When you deploy the CSR 1000V router in the cloud, every branch office, campus, and data center location can access the cloud service directly and securely. This design reduces latency and eliminates the expensive dedicated WAN link.

Figure 5-4 illustrates the use of the CSR 1000V as a VPN gateway to extend an enterprise network into cloud providers.

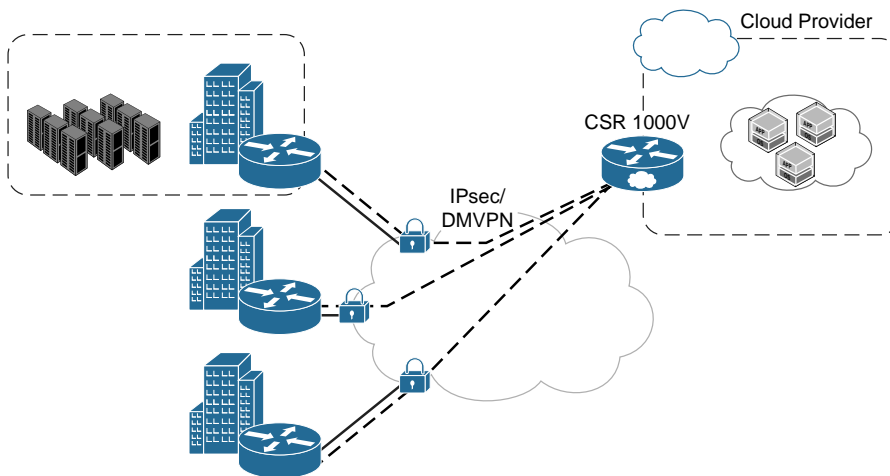


Figure 5-4 *The CSR 1000V as a VPN Service Gateway*

Examples 5-1 and 5-2 show DMVPN hub and spoke configuration examples, respectively.

Example 5-1 *Configuration Example: CSR 1000V as a DMVPN Hub*

```

hostname Hub-1
!
crypto ikev2 keyring VPNKey
peer WANVPN
  address 10.10.0.0 255.255.0.0
  pre-shared-key cisco123
!
crypto ikev2 profile VPN-PROFILE
match identity remote address 10.10.0.0 255.255.0.0
  authentication remote pre-share
authentication local pre-share
keyring local VPNKey
!
crypto ikev2 dpd 60 25 on-demand
!
crypto ipsec security-association idle-time 120
!
crypto ipsec transform-set AES256 esp-aes 256 esp-sha-hmac
  mode transport
!
crypto ipsec profile DMVPN
set transform-set AES256
  set ikev2-profile VPN-PROFILE
!
interface Tunnel1
description DMVPN
bandwidth 10000
ip address 172.16.1.1 255.255.255.0
no ip redirects
ip mtu 1400
ip nhrp authentication VPN123
ip nhrp map multicast dynamic
ip nhrp network-id 101
ip nhrp holdtime 600
ip nhrp redirect
ip tcp adjust-mss 1360
load-interval 30
tunnel source GigabitEthernet1
tunnel mode gre multipoint
tunnel protection ipsec profile DMVPN
!

```

```

interface GigabitEthernet1
description WAN interface
ip address 10.10.1.2 255.255.255.252
!
interface GigabitEthernet2
description LAN interface
ip address 172.16.1.1 255.255.255.0
!
!
router eigrp WAN
!
address-family ipv4 unicast autonomous-system 100
!
  af-interface default
    passive-interface
  exit-af-interface
!
  af-interface Tunnell
    summary-address 172.16.0.0 255.255.0.0
    no passive-interface
  exit-af-interface
!
  topology base
  exit-af-topology
  network 172.16.0.0 0.0.255.255
exit-address-family
!
end

```

Example 5-2 *Configuration Example: ISR as a DMVPN Spoke*

```

hostname Spoke-1
!
crypto ikev2 keyring VPNKey
peer WANVPN
  address 10.10.0.0 255.255.0.0
  pre-shared-key cisco123
!
crypto ikev2 profile VPN-PROFILE
match identity remote address 10.10.0.0 255.255.0.0
  authentication remote pre-share
authentication local pre-share
keyring local VPNKey
!

```

```
crypto ikev2 dpd 60 25 on-demand
!
crypto ipsec security-association idle-time 120
!
crypto ipsec transform-set AES256 esp-aes 256 esp-sha-hmac
  mode transport
!
crypto ipsec profile DMVPN
set transform-set AES256
  set ikev2-profile VPN-PROFILE
!
interface Tunnel1
description DMVPN
bandwidth 10000
ip address 172.16.1.10 255.255.255.0
no ip redirects
ip mtu 1400
ip nhrp authentication VPN123
ip nhrp map multicast 10.10.1.2
ip nhrp map 172.16.1.1 10.10.1.2
ip nhrp network-id 101
ip nhrp holdtime 600
ip nhrp shortcut
ip tcp adjust-mss 1360
load-interval 30
tunnel source GigabitEthernet1
tunnel mode gre multipoint
tunnel protection ipsec profile DMVPN
!
interface GigabitEthernet1
description WAN interface
ip address 10.10.2.2 255.255.255.252
!
interface GigabitEthernet2
description LAN interface
ip address 172.16.2.1 255.255.255.0
!
!
router eigrp WAN
!
address-family ipv4 unicast autonomous-system 100
!
  af-interface default
  passive-interface
  exit-af-interface
```

```

!
af-interface Tunnel1
  summary-address 172.16.0.0 255.255.0.0
  no passive-interface
exit-af-interface
!
topology base
exit-af-topology
network 172.16.0.0 0.0.255.255
exit-address-family
!
end

```

CSR for Secure Inter-Cloud Connectivity

Enterprise cloud consumers may have limitations on communication for a virtual private cloud between different regions that is under the control of the enterprise, making multi-region deployment challenging. With a CSR 1000V virtual router instance running in every virtual private cloud region interconnected through a VPN, you can create a secure network that spans the globe. Figure 5-5 shows a CSR 1000V virtual router connecting multiple locations with enterprisewide VPN service.

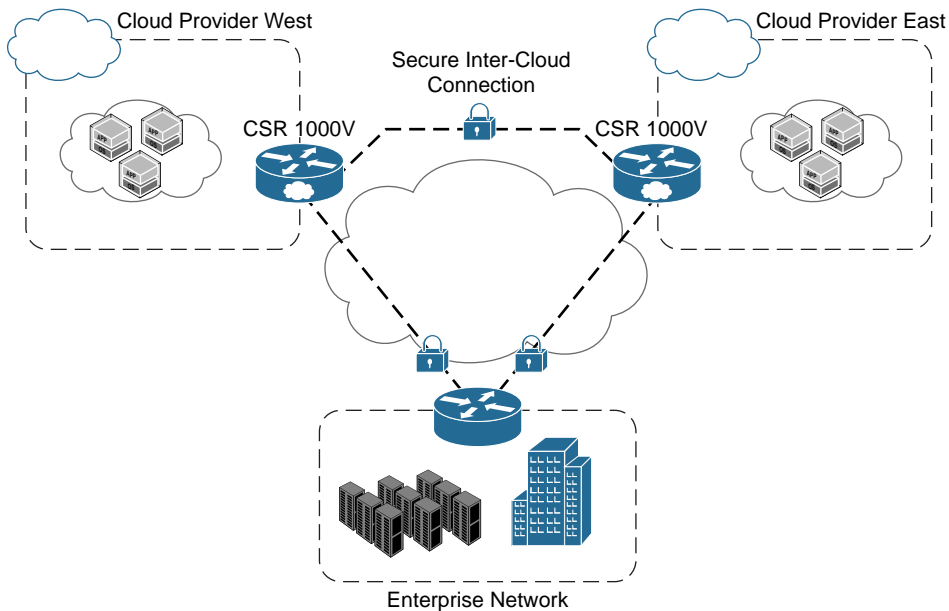


Figure 5-5 CSR 1000V for Secure Inter-regional Cloud Connectivity

Remote VPN Access into the Cloud

The CSR 1000V offers secure remote VPN access for administrative connection to the virtual servers in the cloud (see Figure 5-6). A CSR virtual router offers an SSL VPN gateway solution that allows remote user access to corporate data resources and empowers employees to work from anywhere on corporate laptops as well as personal mobile devices. In addition, the CSR 1000V offers FlexVPN with the Cisco AnyConnect Secure Mobility Client to offer a more robust remote access VPN solution. FlexVPN is a consolidation of multiple VPN frameworks, such as crypto maps and Virtual Tunnel Interface (VTI) into a common set of command-line interface (CLI) configurations. It uses Internet Key Exchange version 2 (IKEv2) as the default protocol for more secure protocol negotiation. FlexVPN running on the CSR 1000V, when combined with AnyConnect Secure Mobility Client, offers end users secure access to resources inside the cloud network from any Internet-enabled location. Example 5-3 shows a sample configuration of this scenario.

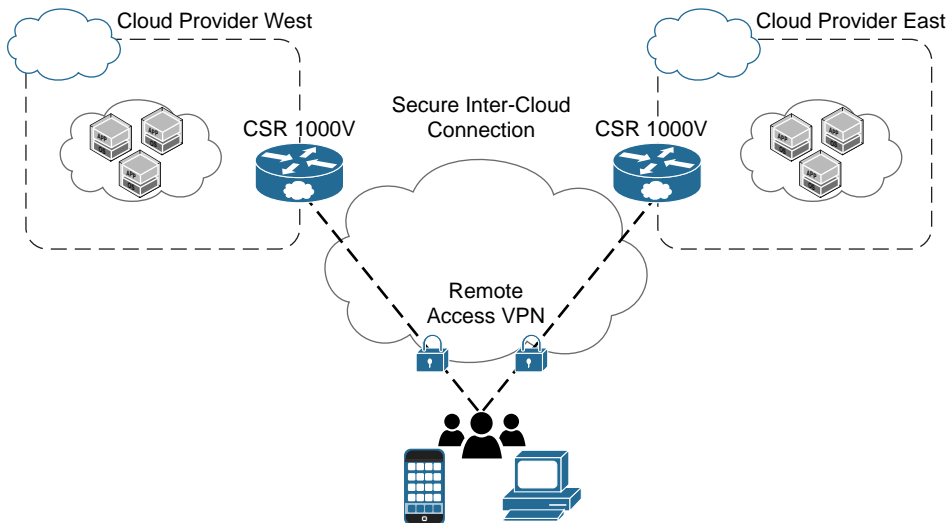


Figure 5-6 CSR for Remote VPN Access into the Cloud

Example 5-3 Configuration Example: CSR as a Remote Access VPN Server with an AnyConnect Client

```
hostname CSR-RA-VPN-Gateway
!
aaa new-model
!
!
radius server FLEXVPN-RADIUS
 address ipv4 10.10.1.21 auth-port 1645 acct-port 1646
 key 7 01300F175804575D72
```

```

!
aaa authentication login FLEXVPN-AAA-LIST group radius
aaa authorization network FLEXVPN-AAA-LIST local
!
clock timezone EST -5 0
clock calendar-valid
!
! Generate an RSA Key with key length of 2048 bytes
crypto key generate rsa general-keys label FLEXVPN-KEY modulus 2048
!
ip domain name cisco.com
!
!Enable IOS CA server for local certificate enrollment
crypto pki server CA
  no database archive
  grant auto
  hash sha256
  eku server-auth client-auth
  no shutdown
!
! Enable HTTP server for certificate enrollment over SCEP
ip http server
!
!Creating a certificate for use of FLEXVPN with AnnyConnect Client
!to support Extended Key Usage (EKU) Requirement.
crypto pki trustpoint FLEXVPN
  enrollment url http://10.10.1.1:80
  fqdn FLEXVPN-HUB.cisco.com
  ip-address none
  subject-name CN=FLEXVPN-GWY.CISCO.COM, OU=IT, O=CISCO
  revocation-check none
  rsakeypair FLEXVPN-KEY 2048 2048
  eku request server-auth client-auth
!
crypto pki authenticate FLEXVPN
!
crypto pki trustpoint CA
  revocation-check crl
  rsakeypair CA
!
!
crypto pki certificate chain FLEXVPN
crypto pki certificate chain CA
  certificate ca 01
!

```

```

!Create FLEXVPN local authorization policy on the router with parameters
!to push out to the clients
crypto ikev2 authorization policy FLEXVPN-POLICY
  pool FLEXVPN-CLIENT-POOL
  dns 10.0.0.120
  netmask 255.255.255.0
  def-domain cisco.com
!
!Configure IKEv2 profile that defines the authentication and authorization
!method and the trustpoint to be used during VPN negotiation
crypto ikev2 profile FLEXVPN-PROFILE
  match identity remote key-id cisco.com
  identity local dn
  authentication remote eap query-identity
  authentication local rsa-sig
  pki trustpoint FLEXVPN
  dpd 60 2 on-demand
  aaa authentication eap FLEXVPN-AAA-LIST
  aaa authorization group eap list FLEXVPN-AAA-LIST FLEXVPN-POLICY
  virtual-template 10
!
!Create IPsec profile that links back to the IKEv2 profile for FLEXVPN
crypto ipsec profile FLEXVPN-IPSEC-PROFILE
  set ikev2-profile FLEXVPN-PROFILE
!
!Define the virtual template from which the VPN session will be using
!and tie the interface to IPsec encryption profile
interface Virtual-Template10 type tunnel
  ip unnumbered GigabitEthernet1
  tunnel mode ipsec ipv4
  tunnel protection ipsec profile FLEXVPN-IPSEC-PROFILE
!
interface GigabitEthernet1
  ip address 10.10.1.1 255.255.255.0
  negotiation auto
!
ip local pool FLEXVPN-CLIENT-POOL 172.16.10.2 172.16.10.254!

```

BGP Route Reflector Use Case for the CSR

Border Gateway Protocol (BGP) is an exterior gateway protocol. BGP is designed to exchange routing and reachability information among autonomous systems (AS) on the Internet or in a large enterprise network. BGP is used to carry large routing tables. All BGP speakers within a single AS must be fully meshed for routing information to be

available on all routers within that AS. There are multiple features and design options available to reduce this requirement of full mesh, and BGP route reflector is one that is prevalent in many deployments.

The route reflector function is a part of Cisco IOS but often requires a dedicated hardware router for optimal scale. On the route reflectors, the internal BGP (iBGP) loop-prevention rule is relaxed for these routers, and they are allowed to re-advertise or reflect routes from one iBGP speaker to another iBGP speaker. The route reflector is a control plane function running on the CSR and requires very minimal data-plane function for sending routing updates to the BGP route-reflector clients. The following rules for route propagation are applied to a route reflector:

- Routes received from an iBGP peer (not a part of the route reflector client), locally generated routes, and routes received from external BGP (eBGP) neighbors are selected as best routes.
- The route received from the route reflector client that is selected as the best route is propagated to all peers.

There are several reasons BGP route reflector design is gaining popularity. BGP is increasingly being deployed in large networks. The need for BGP has also increased due to the deployment of MPLS Layer 3 VPNs. The concept of autonomous systems inheriting the functionality of Layer 3 VPN virtual routing instances between provider edge routers has increased the usage of BGP deployment within a single AS, and route reflectors are leveraged to simplify BGP deployments. The use of route reflectors in this design is mainly to handle the control plane for the BGP routing table. The CSR 1000V offers a cost-effective way to scale the route reflector functionality and handle the BGP control plane, removing the need to deploy dedicated route reflector hardware for this functionality. The actual number of sessions that a route reflector can service depends on a number of factors, such as the number of routes per session, the use of peer groups, the CPU power, and the memory resources of the route reflector.

Table 5-1 provides a comparison between the CSR 1000V and Route Processor 2 (RP2) and shows that the two are similar in terms of handling the route reflector function.

Table 5-1 *CSR 1000V and RP2 Scalability Comparison*

| | CSR 1000V (8GB) | CSR 1000V (16GB) | RP2 (8GB) | RP2 (16GB) |
|------------------------|------------------------|-------------------------|------------------|-------------------|
| IPv4 routes | 8.5MB | 24.8MB | 8MB | 24MB |
| VPNv4 routes | 8.1MB | 23.9MB | 7MB | 18MB |
| Number of BGP sessions | 4000 | 4000 | 8000 | 8000 |

Network Functions Virtualization (NFV) is a cost-effective method to provide virtual route reflector (vRR) functionality compared to traditional deployment using physical hardware. The important thing to keep in mind is to ensure that the deployment of CSR

1000V as a route reflector is not in the data path; you can thereby provide cost-effective capability to manage the BGP control plane that can scale the number of routes in the BGP RIB. Table 5-1 should be used as a mere reference point since these numbers can change based on software release cycle and enhancement in the BGP protocol. Any use case of the CSR 1000V to replace hardware functionality in the design should be done with careful tuning of the hypervisor environment, as discussed in Chapter 3, “Hypervisor Considerations for the CSR.” By default, the IOS XE uses about 50% of the memory for the IOSd process and the remaining memory for other IOS XE processes (valid for RPs with 4GB of memory). Because BGP route reflector is a control plane functionality that runs on the IOSd process only, by default out of 4GB memory available to the CSR 1000V, only 50% of the memory (2GB) will be used for route reflector functionality. To optimize this functionality in the CSR 1000V and to enhance the scalability of the route reflector function, the CSR 1000V with a route reflector license uses the entire 4GB of memory to run the IOSd. The route reflector license available for the CSR 1000V increases its scalability as a route reflector.

The CSR 1000V in a Hierarchical Route Reflector Use Case

Using a hierarchical route reflector concept enhances route reflector design scalability. The cluster concept enables a network admin to add groups of clients to a single route reflector. The router ID of the route reflector identifies the cluster. To increase redundancy and avoid single points of failure, a cluster can have more than one route reflector. The cluster ID is a 4-byte field used to identify the routes and updates within the same cluster or within the iBGP domain.

The concept of cluster ID and hierarchy is used in Figure 5-7 to achieve route reflector redundancy, localization of policies, and route updates within the iBGP domain.

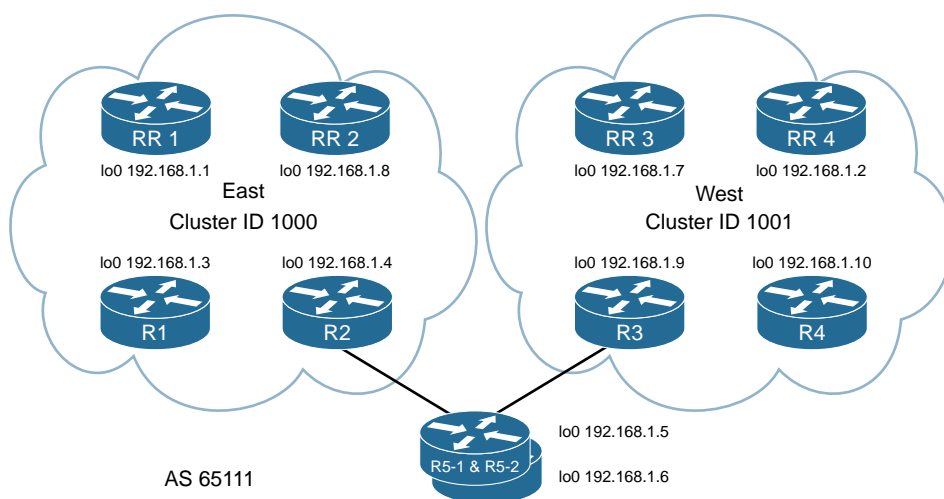


Figure 5-7 Hierarchical Route Reflector Use Case with the CSR 1000V

In this example, AS 65111 is split into two domains: East (cluster ID 1000) and West (cluster ID 1001). Each subdomain has its own route reflectors for the local route reflector clients. The CSR 1000V as a route reflector fully utilizes the hardware resource for the control plane. Very minimum data plane traffic flows between the route reflectors. The following setup illustrates hierarchical route reflector design:

- R1 and R2 are route reflector clients for RR1 and RR2.
- R3 and R4 are route reflector clients for RR3 and RR4.
- R5-1 and R5-2 are conventional routers and form iBGP relationship with the route reflectors in the East and West domains.

The configurations for this route reflector design are listed in Examples 5-4 through 5-14.

Example 5-4 *RR1 Configuration*

```
interface Loopback0
 ip address 192.168.1.1 255.255.255.255
router bgp 65111
  bgp cluster-id 1000
  bgp log-neighbor-changes
  neighbor RR-East peer-group
  neighbor RR-East remote-as 65111
  neighbor RR-East update-source Loopback0
  neighbor RR-East route-reflector-client
  neighbor IBGP peer-group
  neighbor IBGP remote-as 65111
  neighbor IBGP update-source Loopback0
  neighbor 192.168.1.2 peer-group IBGP
  neighbor 192.168.1.3 peer-group RR-East
  neighbor 192.168.1.4 peer-group RR-East
  neighbor 192.168.1.5 peer-group IBGP
  neighbor 192.168.1.6 peer-group IBGP
  neighbor 192.168.1.7 peer-group IBGP
  neighbor 192.168.1.8 peer-group IBGP
```

Example 5-5 *RR2 Configuration*

```
interface Loopback0
 ip address 192.168.1.8 255.255.255.255
router bgp 65111
  bgp cluster-id 1001
  bgp log-neighbor-changes
  neighbor RR-West peer-group
  neighbor RR-West remote-as 65111
```

```

neighbor RR-West update-source Loopback0
neighbor RR-West route-reflector-client
neighbor IBGP peer-group
neighbor IBGP remote-as 65111
neighbor IBGP update-source Loopback0
neighbor 192.168.1.1 peer-group IBGP
neighbor 192.168.1.2 peer-group IBGP
neighbor 192.168.1.5 peer-group IBGP
neighbor 192.168.1.6 peer-group IBGP
neighbor 192.168.1.7 peer-group IBGP
neighbor 192.168.1.9 peer-group RR-West
neighbor 192.168.1.10 peer-group RR-West

```

Example 5-6 R2 Configuration

```

interface Loopback0
 ip address 192.168.1.3 255.255.255.255
router bgp 65111
 bgp log-neighbor-changes
 network 192.168.40.40 mask 255.255.255.255
 neighbor 192.168.1.1 remote-as 65111
 neighbor 192.168.1.1 update-source Loopback0
 neighbor 192.168.1.1 route-reflector-client
 neighbor 192.168.1.2 remote-as 65111
 neighbor 192.168.1.2 update-source Loopback0
 neighbor 192.168.1.2 route-reflector-client

```

Example 5-7 R1 Configuration

```

interface Loopback0
 ip address 192.168.1.4 255.255.255.255
router bgp 65111
 bgp log-neighbor-changes
 neighbor 192.168.1.1 remote-as 65111
 neighbor 192.168.1.1 update-source Loopback0
 neighbor 192.168.1.1 route-reflector-client
 neighbor 192.168.1.2 remote-as 65111
 neighbor 192.168.1.2 update-source Loopback0
 neighbor 192.168.1.2 route-reflector-client

```

Example 5-8 *R5-1 Configuration*

```

interface Loopback0
 ip address 192.168.1.5 255.255.255.255
router bgp 65111
  bgp log-neighbor-changes
  neighbor IBGP peer-group
  neighbor IBGP remote-as 65111
  neighbor IBGP update-source Loopback0
  neighbor 192.168.1.1 peer-group IBGP
  neighbor 192.168.1.2 peer-group IBGP

```

Example 5-9 *R5-2 Configuration*

```

Interface Loopback0
 ip address 192.168.1.6 255.255.255.255
router bgp 65111
  bgp log-neighbor-changes
  neighbor IBGP peer-group
  neighbor IBGP remote-as 65111
  neighbor IBGP update-source Loopback0
  neighbor 192.168.1.1 peer-group IBGP
  neighbor 192.168.1.2 peer-group IBGP

```

Example 5-10 *RR-3 Configuration*

```

interface Loopback0
 ip address 192.168.1.7 255.255.255.255
router bgp 65111
  bgp cluster-id 1001
  bgp log-neighbor-changes
  neighbor RR-West peer-group
  neighbor RR-West remote-as 65111
  neighbor RR-West update-source Loopback0
  neighbor RR-West route-reflector-client
  neighbor IBGP peer-group
  neighbor IBGP remote-as 65111
  neighbor IBGP update-source Loopback0
  neighbor 192.168.1.1 peer-group IBGP
  neighbor 192.168.1.2 peer-group IBGP
  neighbor 192.168.1.5 peer-group IBGP
  neighbor 192.168.1.6 peer-group IBGP
  neighbor 192.168.1.8 peer-group IBGP
  neighbor 192.168.1.9 peer-group RR-West
  neighbor 192.168.1.10 peer-group RR-West

```

Example 5-11 *RR-4 Configuration*

```
interface Loopback0
 ip address 192.168.1.2 255.255.255.255
router bgp 65111
 bgp cluster-id 1000
 bgp log-neighbor-changes
 neighbor RR-East peer-group
 neighbor RR-East remote-as 65111
 neighbor RR-East update-source Loopback0
 neighbor RR-East route-reflector-client
 neighbor IBGP peer-group
 neighbor IBGP remote-as 65111
 neighbor IBGP update-source Loopback0
 neighbor 192.168.1.1 peer-group IBGP
 neighbor 192.168.1.3 peer-group RR-East
 neighbor 192.168.1.4 peer-group RR-East
 neighbor 192.168.1.5 peer-group IBGP
 neighbor 192.168.1.6 peer-group IBGP
 neighbor 192.168.1.7 peer-group IBGP
 neighbor 192.168.1.8 peer-group IBGP
```

Example 5-12 *R3 Configuration*

```
interface Loopback0
 ip address 192.168.1.9 255.255.255.255
router bgp 65111
 bgp log-neighbor-changes
 neighbor 192.168.1.7 remote-as 65111
 neighbor 192.168.1.7 update-source Loopback0
 neighbor 192.168.1.8 remote-as 65111
 neighbor 192.168.1.8 update-source Loopback0
```

Example 5-13 *R4 Configuration*

```
interface Loopback0
 ip address 192.168.1.10 255.255.255.255
router bgp 65111
 bgp log-neighbor-changes
 neighbor 192.168.1.7 remote-as 65111
 neighbor 192.168.1.7 update-source Loopback0
 neighbor 192.168.1.8 remote-as 65111
 neighbor 192.168.1.8 update-source Loopback0
```

Example 5-14 *Snapshot of BGP Update at R4*

```

r57#sh ip bgp 192.168.40.40 255.255.255.255
BGP routing table entry for 192.168.40.40/32, version 3
Paths: (2 available, best #2, table default)
  Not advertised to any peer
  Refresh Epoch 1
Local
  192.168.1.3 (metric 41) from 192.168.1.8 (192.168.1.8)
    Origin IGP, metric 0, localpref 100, valid, internal
    Originator: 192.168.40.40, Cluster list: 0.0.3.233, 0.0.3.232
    rx pathid: 0, tx pathid: 0
  Refresh Epoch 1
Local
  192.168.1.3 (metric 41) from 192.168.1.7 (192.168.1.7)
    Origin IGP, metric 0, localpref 100, valid, internal, best
    Originator: 192.168.40.40, Cluster list: 0.0.3.233, 0.0.3.232
    rx pathid: 0, tx pathid: 0x0

```

The highlighted cluster ID in Example 5-14 shows the path of origination of the route in the BGP AS.

Planning for Future Branch Design with the CSR 1000V

This section provides an overview of the enterprise branch router and its evolution from a traditional hardware branch router. You will also learn about the functionality of an integrated router that uses NFV elements and its evolution.

The traditional network components that impact the branch design are routing, firewall, and encryption. Today, the network is being converged as a platform to launch IT services. As this trend continues, branch solutions will need to scale these multiple technology domains, moving a branch from a single node in a high-availability domain to a multiple-node environment.

In Figure 5-8, notice that the branch router's role is more than just a normal routing functionality. The transition in basic routing is seen in the introduction of performance-based routing (Cisco Performance Routing [PfR]), where the routing concept has evolved from prefix-based routing to more intelligent path selection. Performance-based routing allows the router to make an intelligent path decision based on application service level requirements.

The current WAN branch router supports VPN, voice, WAN optimization engine, a router-based firewall, integrated switches, and UCS computing blades. All these elements are added to the router as features in the IOS code and interface cards. The rapid deployment of these features has driven the need for using services outside the device. Leveraging computing infrastructure present in the router to spawn network virtual devices reduces cost factors such as power, space, and cabling.

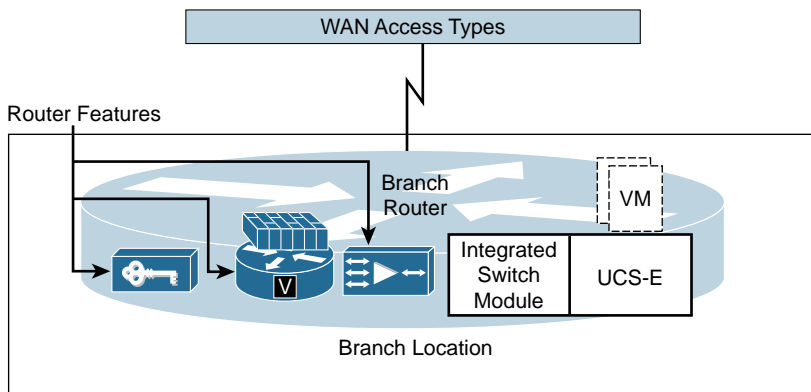


Figure 5-8 *Current Branch Router Capabilities*

By using NFV, you can spawn virtual devices to scale to new feature requirements. In Figure 5-9, the branch router has a security gateway (firewall and Sourcefire) that provides functionalities such as firewall services, Advanced Malware Protection (AMP), Application Visibility and Control (AVC), and URL filtering. Instead of using firewall functionality in the router, you have an option of using an NFV element that provides additional security functionality.

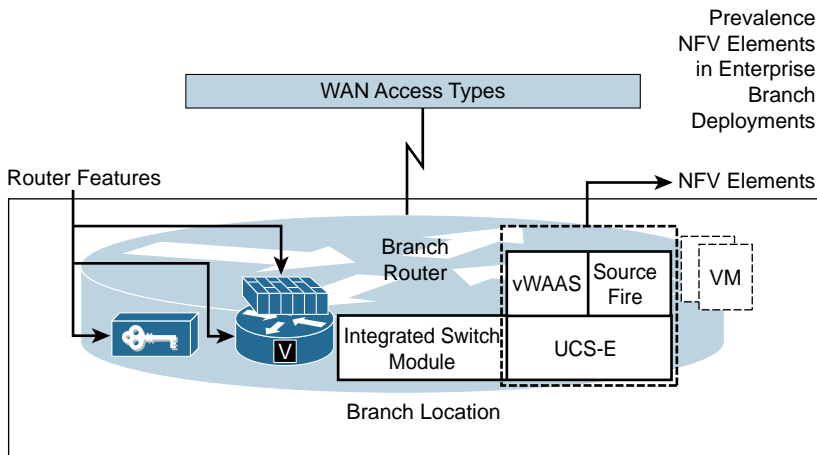


Figure 5-9 *Adoption of NFV Elements*

The introduction of computing as an integrated router platform provides a new paradigm for launching services and NFV from this platform. The UCS E-Series blade server that runs in ISR has two internal Gigabit Ethernet interfaces connecting to the router and at least one external Gigabit Ethernet interface:

- One of the internal interfaces is a Layer 2 interface that connects the server blade directly to the backplane switch for traffic destined to the virtual server. The other internal interface is a Layer 3 interface that connects the blade to the Cisco

ISR route engine for management traffic such as Cisco Integrated Management Controller (CIMC) and the host operating system configuration.

- The external Gigabit Ethernet port takes care of external connection use cases. (One of them is WAN connectivity, or connectivity to the external firewall.)

The UCS E-Series uses a bare metal hypervisor platform that is a joint Cisco and VMware solution to create a virtualization-ready platform. The ESXi is optimized to function on the UCS E blade server hardware and is the beginning of a new thought in designing an NFV solution for the branch. These are some of the benefits of having an integrated computing platform in the routers:

- Virtual servers can be provisioned more quickly than their physical counterparts.
- Virtual servers require less space, power, and cooling.
- The integrated solution design is easy to operate within a single domain.

Evolution of Branch Virtualization

You should know by now that the branch router plays a more vital role than just a normal routing functionality and adds services to the branch for taking care of IT requirements. This evolution is a cost-effective consolidation of services within a domain. The evolution of orchestration and management for NFV technology helps network architects leverage NFV technology to meet the same network requirements within a single box rather than using multiple dedicated appliances. The use of the CSR 1000V removes the need for dedicated routing hardware, and a complete suite of NFV elements can replace IT service functionality needed at the branch.

The NFV approach to branch virtualization opens up new technology avenues by providing a platform for customers to deploy virtualized network elements as required. Coupling this with an easy-to-use end-to-end orchestration and management framework, enterprises are able to significantly reduce costs and get better return on investment (ROI) by avoiding expensive truck rolls to enable services at their branches. These are the key aspects of branch virtualization:

- **Programmability**—You can leverage open APIs to enable better automation of network services while improving visibility.
- **Agility**—You gain flexibility in deploying services quickly in a timely manner. You can improve business efficiency in capital and operations by meeting the evolving business requirements, including traffic growth, diversity of traffic types, performance, reliability demands, and expectations.
- **Simplicity**—You can reduce complexity from services and operations and endorse more nimble business models. You gain the ability to manage all branches with a single pane of glass.

Branch virtualization leverages a specialized platform customized to take care of NFV requirements and offload special functions, such as encryption and customized drivers, to provide increased performance for different NFV elements. You will see the terms NFV and VNF (Virtual Network Functions) throughout this chapter. It is good to understand the difference between the two terminologies. NFV is a complete virtual service paradigm, while VNF is a virtual network element or service that is part of the NFV framework. These are the foundation blocks for this next-generation networking gear:

- Customized x86 hardware to host VNF elements
- Optimized hypervisor platform to launch VNF elements
- Solid foundation of orchestration engine
- Flexible options for I/O

Figure 5-10 shows an optimized hypervisor with special drivers for network services (aligned to the hardware platform) that facilitates performance throughput for NFV elements. The hardware platform also takes care of special functions like crypto acceleration to provide increased performance.

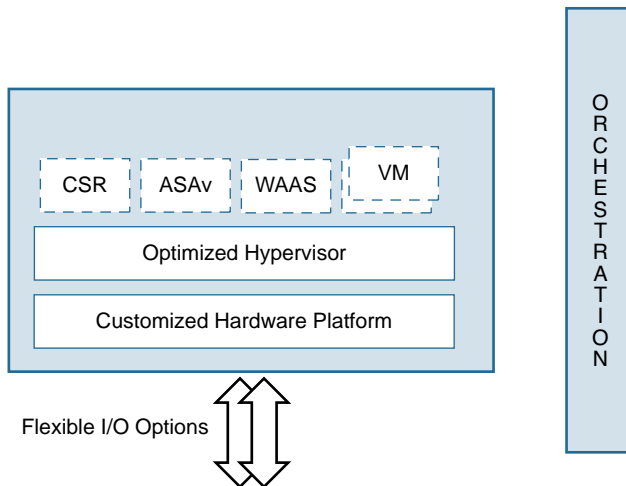


Figure 5-10 *High-Level Conceptual Block: Next-Generation Branch*

An internal virtual switch provides connectivity between VMs or VNFs and is used to switch traffic between service elements. This internal path offers optimal traffic forwarding between the VNF elements running inside the platform. Availability of different types of external I/O interfaces allow the deployment of this device in different use cases.

One of the challenges in offering new services at a branch is that connecting up the service chain and deploying new applications takes a great deal of time and effort. It means racking and stacking network devices and cabling them together in the required

sequence. Each new service requires a specialized hardware appliance that has to be individually configured. The chances for errors are high, and a problem in one component could disrupt the entire network.

NFV enables on-demand service and centralized orchestration for integrating the new service into the existing ones—in essence creating a service chain. For example, a customer who desires firewall functionality can use a portal to choose among a list of VNFs (ASA, vWAAS, and so on), which will then be deployed dynamically on the platform. Enterprises gain the ability to choose “best of breed” VNFs to implement a particular service.

When scaling virtual network functions, it is important to consider the performance factor. The hardware drivers will be available only on limited VNF devices to optimize their performance. These drivers fit within the hypervisor code, enabling it to interact with the I/O function. Figure 5-10 calls this *optimized hypervisor*. The branch virtualization platform has hardware-assisted binding to create and configure the hardware path for the flow in the data plane between the VNF elements. The future branch design with NFV will be similar to the high-level concept of ISR design, where a single box takes care of multiple functions.

Virtual network services need to have simple management and orchestration. These services leverage an auto-provisioning agent (for zero-touch deployment) that connects to the server in the cloud. Zero-touch provisioning plays an important role in a software-defined WAN (SD-WAN) framework for VNF elements. This offers the capability for a centralized controller to manage a multitude of VNF elements at scale.

As shown in Figure 5-10, the branch virtualization is not complete without a good orchestration solution. Some of the key characteristics of the orchestration solution are to provide role-based access to admins, ability to instantiate new VNF elements, and configuration management from a centralized management tool. The use of tools like Cisco Network Service Orchestrator enabled by Tail-f improves the management and deployment of VNF elements.

The network service orchestrator shown in Figure 5-11 contains a service manager component and a device manager component. The service manager maintains information on different domain components required for the service, and the device manager keeps the configuration of each device type. The network element drivers provide several options that can leverage NETCONF, CLI, SNMP, and REST, thereby providing a multitude of automation coverage.

It is important to understand the new phase of orchestration using NETCONF and YANG. A detailed use case of NETCONF and YANG with Tail-f is provided in Chapter 8, “CSR 1000V Automation, Orchestration, and Troubleshooting.” NETCONF is a protocol that is defined by the IETF for installing, modifying, or deleting configuration from network devices. One can argue that SNMP also does the same functionality on a network device. The reasons for the new NETCONF protocol were lack of a defined discovery process, nondefined framework to get the MIB, UDP-based limitation, and standard security mechanism. NETCONF uses a structured layering concept, as shown in Figure 5-12.

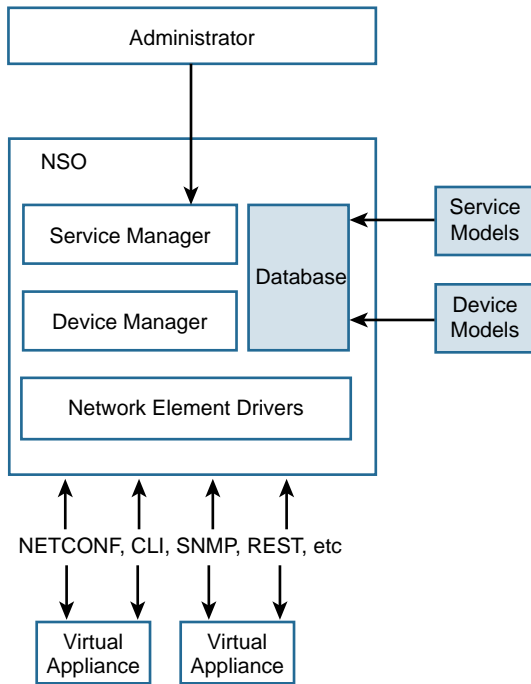


Figure 5-11 Tail-F Framework

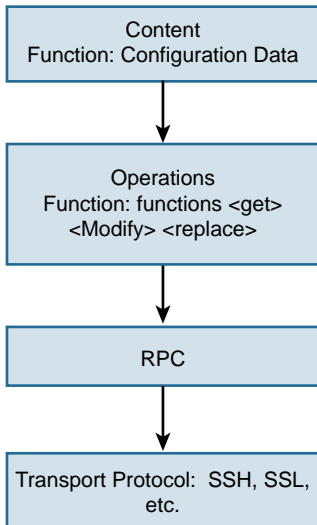


Figure 5-12 NETCONF Framework

This layering concept provides a structured approach for configuration templates used in user-defined operations to get or modify the configuration. The delivery method used here is RPC. An RPC message gets transported using different protocols, such as SSL, to reach the network device. This layered approach provides the flexibility needed for multiple use cases.

The YANG data modeling language is used to create user-defined functions for the data modeling in the NETCONF model. The configuration data in the top layer is modeled by YANG. The YANG model provides easy representation that can be read by anyone, a hierarchical configuration model that can be scaled and reused based on user needs, and a protocol supporting the underlay RPC operations. With the rapid adoption of NETCONF, the modeling language YANG that provides a simple, structured, and scalable approach for modeling the data has a promising future in the field of network orchestration.

This next-generation solution framework will provide the following deployment advantages:

- It will lead to increased adoption of network functional virtualization.
- It will simplify deployment of IT services in branches.
- It will align network services with the virtual computing and network domains.
- It will leverage the x86-based platform.
- It will consolidate multiple domains into a single network appliance.
- It will bring integrated switching capability with orchestration tools.
- It will provide agility in deploying new services and making changes to existing services.
- Enterprises will be able to deploy best-of-breed virtual network services from different vendors.

LISP and CSR

Locator/ID Separation Protocol (LISP) is a schema for IP addressing and routing that essentially separates the endpoint address space from the routing locator. To understand LISP and the issues it tries to address, it is appropriate to draw a metaphor to DNS. DNS, like IP routing, has to deal with a large number of database entries. With DNS, the end user has a name that it tries to resolve to reach a particular endpoint. DNS servers within an authoritative zone make this resolution possible for you. However, all DNS entries may not be present on the local DNS server. If the entry an end user is looking for is not present on the local DNS, the DNS server queries an upstream server for the information. The idea here is not to have all entries present on all servers at all times but to query and resolve an end point request on demand. Figure 5-13 illustrates the basic concept of DNS.

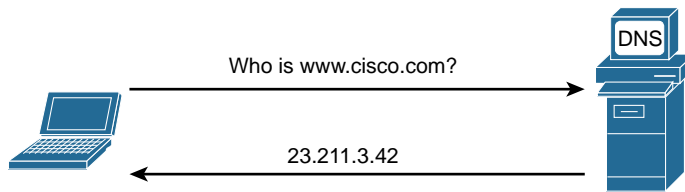


Figure 5-13 *DNS Diagram*

The IP routing entries across the Internet are present on all Internet routers. All routers hold the entire Internet’s routing entries. Summary routes reduce these entries to a certain extent. However, it is still a very large database. LISP tries to address this problem by creating two address spaces. The first address is the routing endpoint identifier, which is the entry the Internet routers use to forward the packet. The second address is the actual endpoint identifier that sits behind the routing endpoint identifier. Figure 5-14 illustrates the basic LISP concept.

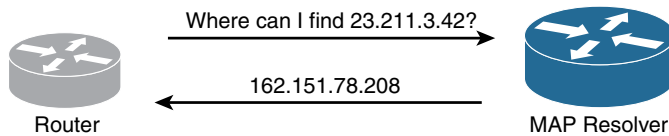


Figure 5-14 *Basic LISP Diagram*

To illustrate using DNS, say you have a router requesting your MAP resolver for an IP address it does not have a routing entry for. The MAP resolver tells the router to send the packet to a “routing endpoint,” which can forward the packet to the end host the router wants to reach. The router then encapsulates the original IP packet to send it to the routing endpoint the MAP resolver provided.

The fact that a MAP server is instructing a router which routing endpoint connects to the end device is reminiscent of IP mobility. An end device can keep its IP address intact and move to a completely different administrative domain. All you need to do is update the MAP resolver for the new routing locator entry for the endpoint! This mobility use case is particularly useful when it comes to managing and moving VMs across administrative domains.

LISP Terminology

Figure 5-15 illustrates IP mobility simplified, using LISP. You can use the scenario illustrated in this figure to become more familiar with LISP terminologies. The router that receives packets from the hosts (in this case, router SJ 162.1.1.5, which receives packets from host 15.1.1.2) to be sent to remote LISP sites is called the ingress tunnel router (ITR). The ITR encapsulates packets for remote LISP sites; for non-LISP sites, it just forwards packets natively. The egress tunnel router (ETR) receives LISP packets, decapsulates them, and sends them to the end device. As illustrated in the example in Figure 5-15, the ETR and ITR are usually the same device and can be referred to as xTR . The host or endpoint (15.1.1.2 in Figure 5-15) is called an endpoint identifier (EID).

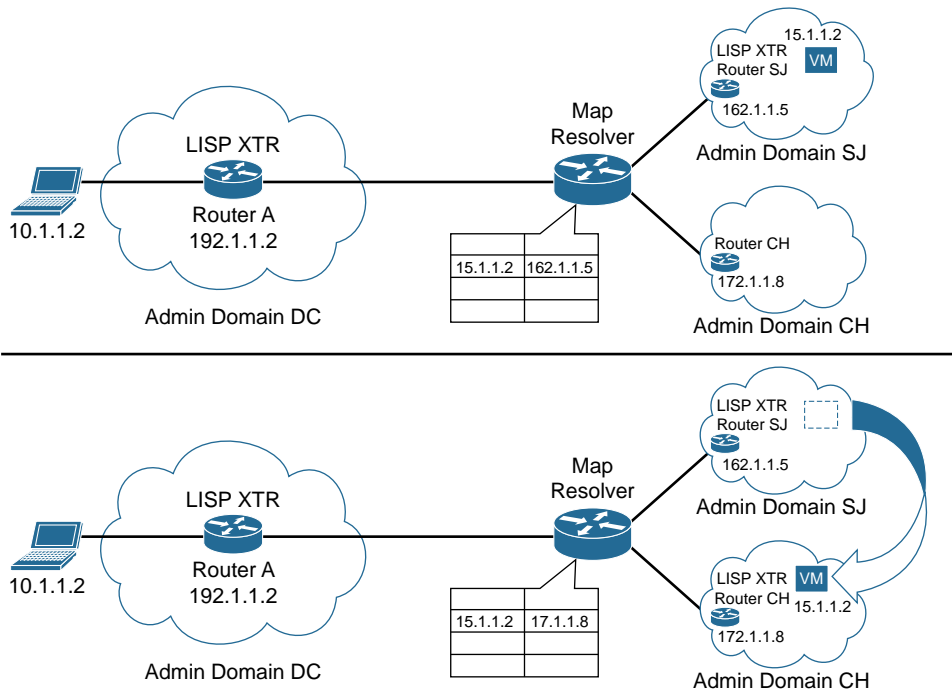


Figure 5-15 Preliminary LISP Use Case (IP Mobility)

The ETR's IP address is called the Routing Locator (RLOC). The RLOC in Figure 5-15 is 162.1.1.5 for administrative domain SJ. The xTR registers this RLOC as the IP address for reaching 15.1.1.2, which is the EID. This registration is stored in the Map Server (MS). The MS keeps a table of this EID-to-RLOC mapping, which it receives from the ETR.

The MAP Resolver (MR) is the server that gives you the RLOC-to-EID mapping. (Customers usually configure the same device as MS and MR.)

An ITR encapsulates traffic for LISP sites and natively forwards traffic for non-LISP sites. However, in certain cases, an ITR may have to encapsulate packets that are coming in from a non-LISP site but are destined for a LISP site. This is a special functionality that an ITR needs to support and is referred to as PITR, where *P* stands for *proxy*. PITR is required to connect non-LISP sites to LISP sites.

Similarly, PETR is a functionality supported by an ETR when it accepts LISP-encapsulated packets from an ITR or PITR for non-LISP sites. The PETR in this case must decapsulate the LISP packet and forward it natively to the non-LISP sites. The PETR is useful when dealing with routers that have Unicast Reverse Path Forwarding (uRPF) configured in strict mode. uRPF drops packets from unknown source addresses. Consider a situation where a host is sending packets from a LISP site to a non-LISP site. Because the LISP EIDs are not advertised, the LISP encapsulated packets have a source address of EIDs that are not known to the routers outside the LISP site domain. Routers

with uRPF configured drop these packets. PETR functionality comes in handy in such a scenario.

Table 5-2 shows a summary of the key LISP abbreviations.

Table 5-2 *Key LISP Abbreviations*

| Abbreviation | Description |
|---------------------|--|
| RLOC | Routing locator |
| ITR | Ingress tunnel router (responsible for encapsulating LISP packets for remote LISP sites) |
| PITR | Proxy ingress tunnel router |
| ETR | Egress tunnel router (responsible for decapsulating LISP packets for local EIDs) |
| PETR | Proxy egress tunnel router |
| xTR | ETR and ITR as the same device |
| PxTR | Proxy xTR, which allows communication between the LISP sites and non-LISP addresses |
| EID | Endpoint identifier |
| MS | Map server EID-to-RLOC mapping |
| MR | Map resolver RLOC-to-EID mapping |

The following sections cover the LISP data plane and control plane flows using Figure 5-15.

The LISP Data Plane

Note in Figure 5-15 that a host 10.1.1.2 within admin domain DC (a LISP site) wants to send a packet to 15.1.1.2 within admin domain CH (another LISP site).

Host 10.1.1.2 sends a packet with a source IP address of 10.1.1.2, which is the EID in domain DC and a destination IP address of 15.1.1.2, which is the EID of the host within domain CH. The host sends this packet to router A, which is the ITR. The router encapsulates this packet in a LISP header after it looks up the RLOC to reach the destination EID. To reach 15.1.1.2, the router needs to use RLOC router CH (172.1.1.8). It then takes the original IP packet (with source IP 10.1.1.2 and destination IP 15.1.1.2) and imposes a UDP header followed by a new IP header with source and destination as the RLOCs (here with source 192.1.1.2 and destination 172.1.1.8).

The LISP Control Plane

Before data plane encapsulation and forwarding can take place, there is an additional lookup involved. The EID-to-RLOC mapping needs to be done in order for the ITR to encapsulate the packet and spurt it over the WAN. Since EIDs can be within a private

address space from RFC 1918, they are not routable over the Internet. So the ITR has the LISP header imposed on it with the routable RLOCs to ensure proper routing over the WAN.

Three kinds of packets can be used to make this mapping possible: data probe, map reply, and map request. A *data probe* is used to send to the MAP server to get an RLOC for an EID. When an authoritative ETR gets this packet (the data probe packet has the inner destination address copied to the outer, which means the outer header carries the EID), it sends a *map reply* to the ITR. The ITR uses this authoritative ETR's IP address as the RLOC IP address and encapsulates the packet and sends it over. An ITR can send a *map request* packet to the map server to get the EID-to-RLOC mapping, too.

Figure 5-16 illustrates the LISP packet, including the inner header, the outer header, and the LISP headers.

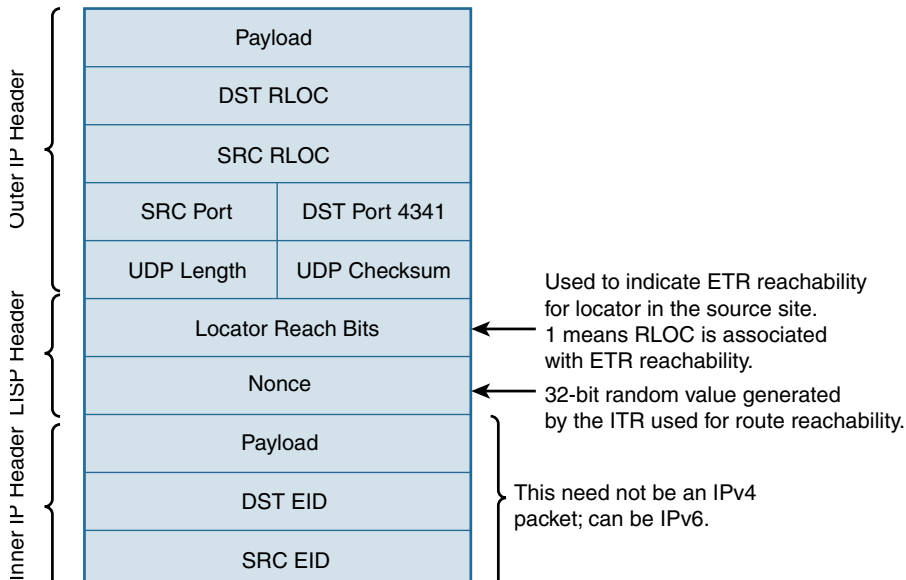
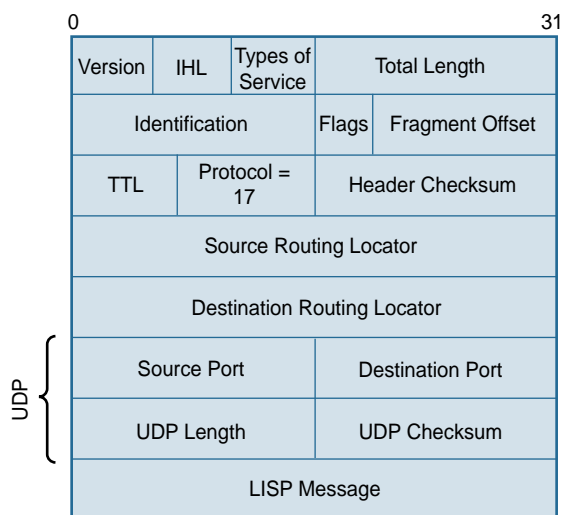


Figure 5-16 LISP Packet Header

Figure 5-17 shows the LISP control plane packet format.



UDP Map-Request: Source port is chosen by the sender; destination port is 4342.

UDP Map-Reply: Source UDP port number is set to 4342. Destination UDP port number is copied from the source port of either the Map-Request or the invoking data packet.

Figure 5-17 *LISP Control Plane Packet*

Figure 5-18 illustrates the LISP map request message format and the special bits in the header and their functions:

- **A**—This is the authoritative bit, set to either of the following:
 - 0 for UDP-based map requests sent by an ITR
 - 1 when an ITR wants the destination site to return the map reply rather than the mapping database system
- **M**—This is the map-data-present bit. When set, it indicates that a map reply record segment is included in the map request.
- **P**—This is the probe bit, which indicates that a map request *should* be treated as a locator reachability probe. The receiver *should* respond with a map reply with the probe bit set, indicating that the map reply is a locator reachability probe reply, with the nonce copied from the map request.
- **S**—This is the solicit-map-request (SMR) bit.
- **p**—This is the PITR bit. It is set to 1 when a PITR sends a map request.
- **s**—This is the SMR-invoked bit. It is set to 1 when an xTR is sending a map request in response to a received SMR-based map request.

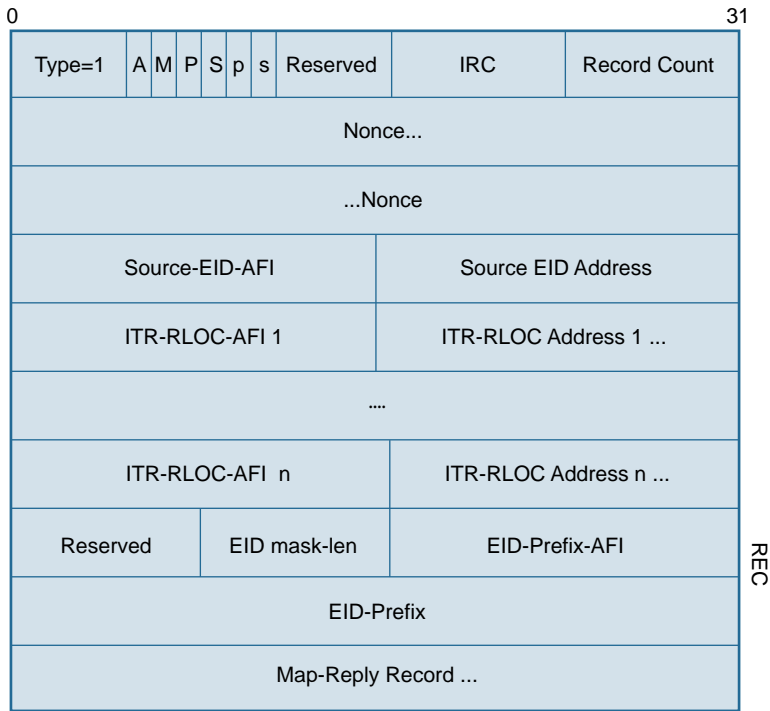


Figure 5-18 LISP Map Request Message Format

Figure 5-19 illustrates the LISP map reply message format.

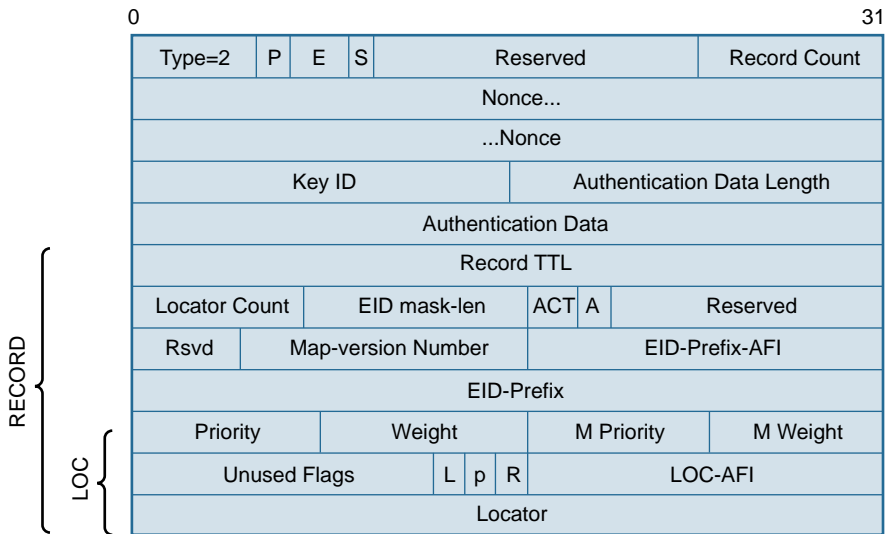


Figure 5-19 LISP Map Reply Message Format

RFC 6830 details the packet field descriptions on different LISP packets, such as map reply.

Typical LISP Use Cases

The following sections examine common LISP use cases, detailing only the network-to-network interconnectivity feature. The configuration for this deployment is included at the end of the chapter.

In the following use cases, the CSR is being used as an MS/MR router, xTR, and PxTR.

IP Mobility

As is evident from the LISP protocol overview in this chapter, LISP encapsulation creates a dynamic control plane that does not require the user to preconfigure the endpoint for mobility. The ITR queries the MS/MR and gets the RLOC dynamically for each EID it needs to reach. This enables the endpoint device to keep its identity and not have a region-based identity. For example, the endpoint can have an IP address within a data center in San Jose and keep the same IP address when moved to a New York data center. All the endpoint needs to do is to update the MS/MR (that is, register itself to the new MS/MR). This is particularly useful in a virtualized data center environment. These days, EIDs are virtual machines. With LISP, they can move to a new data center and retain their IP addresses.

In a cloud environment, to abstract the IP addresses within the cloud, the CSR offers the capability of using LISP as a feature to accomplish this IP mobility function.

IPv6 Migration

LISP supports both IPv4 and IPv6 addressing. The secret is in the way the LISP header is designed. In Figure 5-16, it is evident that it does not matter what the inner IP packet looks like. It can be an IPv6 packet, and LISP will encapsulate it with a UDP header and tunnel it across to the RLOC, where it is decapsulated and forwarded as a native IPv6 packet. This is very useful for sites that want to migrate to IPv6 and still have an IPv4 backbone for transport. LISP is therefore a low-cost IPv6 migration option for enterprises.

Network-to-Network Connectivity

LISP enables the reduction of the routing table size by aggregating networks/hosts behind the RLOC with just the RLOC address. This is ideally suited for network-to-network connectivity. A large network, when connecting to a smaller network, does not need to inject its entire routing table into the smaller network. LISP enables smaller networks to have just one route to reach the LISP gateway within the larger networks. This not only prevents smaller networks from being overwhelmed with large routing tables but also helps the bigger providers encapsulate their TOS value within a UDP packet. This way, the transit networks are unable to modify the TOS value.

In the use case described here, CSR is being used for all the following:

- **LISP-to-MPLS Gateway (LMGW)**—This is the router that terminates the MPLS connection. After the LMGW terminates the MPLS connection, it encapsulates the packet into LISP and sends it to the ASBR. This is for west-to-east traffic in Figure 5-20. For east-to-west traffic, the LMGW decapsulates the LISP packet, tags it with an MPLS label, and puts it on the MPLS core.
- **Map server (MS)**—The MS receives a MAP request from an ITR and finds the corresponding ETR RLOC mapping for the EID.
- **Map resolver (MR)**—The MR receives map requests from the ITR and uses the mapping database to find the corresponding ETRs to answer those requests.
- **Route reflector (RR)**—The RR is a BGP route reflector that reflects BGP routes to route reflector clients within an iBGP network, enabling route learning without requiring full-mesh neighbor adjacency.

Network-to-Network Interconnection Topology and Configuration

This section details the network-to-network interconnection topology and configuration.

Figure 5-20 illustrates the LISP network-to-network connectivity topology.

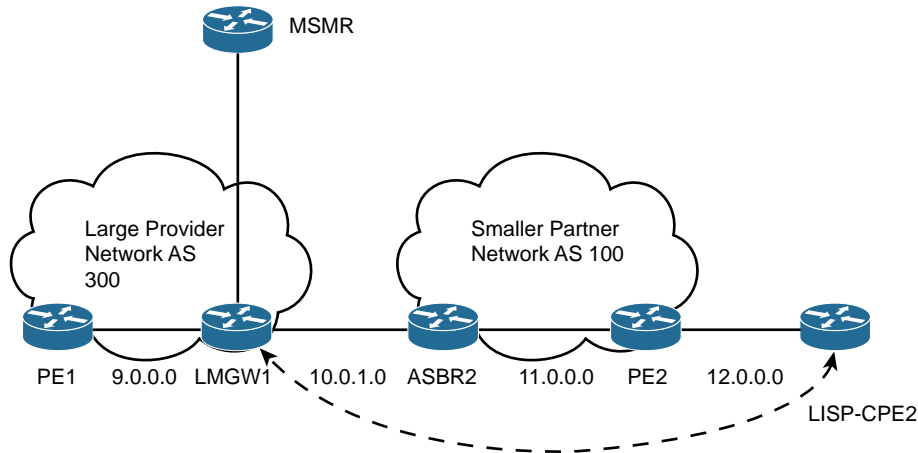


Figure 5-20 LISP Network-to-Network Connectivity Example

Example 5-15 details the LISP-to-MPLS Gateway (LMGW) configuration.

Example 5-15 *LMGW Configuration*

```
vrf definition Trans_blue
  rd 1000:1001
  !
  address-family ipv4
  exit-address-family
  !
vrf definition blue
  rd 1000:1
  route-target export 1000:1
  route-target import 1000:1
  route-target import 1000:1001
  !
  address-family ipv4
  exit-address-family
  !
interface Loopback0
  ip address 102.1.1.1 255.255.255.255
  ip ospf 1 area 0
  !
interface Loopback1
  vrf forwarding Trans_blue
  ip address 10.100.8.2 255.255.255.255
  !
interface Loopback100
  vrf forwarding blue
  ip address 192.168.1.2 255.255.255.255
  !
interface LISP1
  !
interface LISP1.101
  !
interface GigabitEthernet1
  description connected to AVPN PE
  ip address 9.0.0.2 255.255.255.0
  ip ospf 1 area 0
  negotiation auto
  mpls ip
  !
interface GigabitEthernet2
  description connected to AVPN PE
  no ip address
  ip ospf 1 area 0
  negotiation auto
```

```
mpls ip
!
interface GigabitEthernet2.100
  encapsulation dot1Q 100
  vrf forwarding blue
  ip address 10.10.11.1 255.255.255.252
!
interface GigabitEthernet2.101
  encapsulation dot1Q 101
  vrf forwarding Trans_blue
  ip address 10.10.1.1 255.255.255.252
interface GigabitEthernet3
  no ip address
  negotiation auto
!
interface GigabitEthernet3.2
  description connected to Partnet ASBR
  encapsulation dot1Q 2
  vrf forwarding Trans_blue
  ip address 10.0.1.2 255.255.0.0
!
router lisp 1
  locator-table vrf Trans_blue
  eid-table vrf blue instance-id 101
  ipv4 route-import map-cache bgp 200 route-map CUST-EID
  exit
!
no ipv4 map-cache-persistent
ipv4 proxy-etr
ipv4 proxy-itr 10.100.8.2
ipv4 itr map-resolver 10.100.1.2
exit
!
router ospf 1
!
router bgp 200
  bgp log-neighbor-changes
  neighbor 101.1.1.1 remote-as 200
  neighbor 101.1.1.1 description iBGP
  neighbor 101.1.1.1 update-source Loopback0
!
  address-family ipv4
    network 10.0.0.0 mask 255.255.0.0
    network 102.1.1.1 mask 255.255.255.255
  neighbor 101.1.1.1 activate
```

```

neighbor 101.1.1.1 next-hop-self
exit-address-family
!
address-family vpnv4
neighbor 101.1.1.1 activate
neighbor 101.1.1.1 send-community extended
neighbor 101.1.1.1 next-hop-self
exit-address-family
!
address-family ipv4 vrf Trans_blue
redistribute connected
neighbor 10.0.1.1 remote-as 100
neighbor 10.0.1.1 activate
neighbor 10.10.1.2 remote-as 300
neighbor 10.10.1.2 activate
neighbor 10.10.1.2 send-community both
exit-address-family
!
address-family ipv4 vrf blue
neighbor 10.10.11.2 remote-as 300
neighbor 10.10.11.2 activate
neighbor 10.10.11.2 send-community both
exit-address-family
!
ip bgp-community new-format
ip community-list 1 permit 991:1177
no ip http server
no ip http secure-server
ip route 10.0.0.0 255.255.0.0 Null0
!
!
route-map CUST-EID permit 10
match community 1

```

Example 5-16 details the MS/MR configuration for the network-to-network connectivity use case.

Example 5-16 *MS/MR Configuration*

```

vrf definition Trans_blue
rd 1000:1001
!
address-family ipv4
exit-address-family
!

```

```
vrf definition blue
  rd 1000:1
  route-target export 1000:1
  route-target import 1000:1
  route-target import 1000:1001
  !
  address-family ipv4
  exit-address-family
  !
ip vrf CustomerA
  rd 8000:1
  route-target export 8000:101
  route-target import 8000:101
  !
interface Loopback1
  vrf forwarding Trans_blue
  ip address 10.100.1.2 255.255.255.255
  !
interface Loopback100
  ip vrf forwarding CustomerA
  ip address 192.168.1.100 255.255.255.255
  !
interface GigabitEthernet1
  ip address 2.10.38.7 255.255.0.0
  negotiation auto
  !
interface GigabitEthernet2
  no ip address
  negotiation auto
  !
interface GigabitEthernet2.2
  encapsulation dot1Q 100
  vrf forwarding blue
  ip address 10.10.11.2 255.255.255.252
  !
interface GigabitEthernet2.3
  encapsulation dot1Q 101
  vrf forwarding Trans_blue
  ip address 10.10.1.2 255.255.255.252
  !
!
router lisp 1
  locator-table vrf Trans_blue
  eid-table vrf blue instance-id 101
  ipv4 route-export site-registration
```



```
exit
!
site PCE2
  authentication-key cisco123
  eid-prefix instance-id 101 14.1.1.0/24
exit
!
ipv4 map-server
ipv4 map-resolver
exit
!
router bgp 300
  bgp log-neighbor-changes
  redistribute lisp
  !
  address-family ipv4 vrf Trans_blue
    network 10.100.1.2 mask 255.255.255.255
    redistribute connected
    neighbor 10.10.1.1 remote-as 200
    neighbor 10.10.1.1 activate
    neighbor 10.10.1.1 send-community both
  exit-address-family
  !
  address-family ipv4 vrf blue
    redistribute lisp route-map CUST-EID
    neighbor 10.10.11.1 remote-as 200
    neighbor 10.10.11.1 activate
    neighbor 10.10.11.1 send-community both
    neighbor 10.10.11.1 route-map CUST-EID out
    neighbor 10.10.11.1 filter-list 1 in
  exit-address-family
  !
ip route vrf Trans_blue 0.0.0.0 0.0.0.0 10.10.1.1
!
route-map CUST-EID permit 10
  set community 991:1177
```

Example 5-17 details the show commands you need to verify the proper functioning of LISP.

Example 5-17 *show Commands*

```

LMGW1# show run | sec router lisp
router lisp 1
  locator-table vrf Trans_blue
  eid-table vrf blue instance-id 101
  ipv4 route-import map-cache bgp 200 route-map CUST-EID
  exit
!
no ipv4 map-cache-persistent
ipv4 proxy-etr
ipv4 proxy-itr 10.100.8.2
ipv4 itr map-resolver 10.100.1.2
exit

LMGW1# show ip lisp 1 instance-id 101 map-cache
LISP IPv4 Mapping Cache for EID-table vrf blue (IID 101), 1 entries

14.1.1.0/24, uptime: 00:01:37, expires: 23:58:22, via map-reply, complete
  Locator    Uptime      State      Pri/Wgt
  12.0.0.2   00:01:37   up         1/100

LMGW1# show ip route vrf Trans_blue
Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 6 subnets, 3 masks
C       10.0.0.0/16 is directly connected, GigabitEthernet3.2
L       10.0.1.2/32 is directly connected, GigabitEthernet3.2
C       10.10.1.0/30 is directly connected, GigabitEthernet2.101
L       10.10.1.1/32 is directly connected, GigabitEthernet2.101
B       10.100.1.2/32 [20/0] via 10.10.1.2, 01:56:11
C       10.100.8.2/32 is directly connected, Loopback1
      12.0.0.0/24 is subnetted, 1 subnets
B       12.0.0.0 [20/0] via 10.0.1.1, 01:50:19

LMGW1#

```

Summary

With the transition to virtual data centers and cloud services, enterprises are looking for a secure and consistent design to extend the network connection to these locations. With the multitude of security and network segmentation features embedded, the CSR 1000V is in a prime position to tackle these enterprise requirements in the cloud. This chapter has discussed using the CSR to extend data center networks and to secure inter-cloud connectivity, for providing remote VPN access into the cloud, and for dynamic site-to-site VPNs with DMVPN technology.

In addition, this chapter provided a use case for a route reflector with the CSR 1000V. A route reflector is primarily control plane driven, which makes it a perfect use case for the CSR 1000V. You have learned about using the CSR 1000V in the NFV framework and looked at the evolution of the branch architecture that we will soon be seeing. This chapter's LISP with the CSR 1000V use case has helped you understand the importance of IP mobility and segmentation requirements for enterprises in the cloud environment.

This chapter provides an overview of different use case scenarios and the technology associated with the deployment use cases. The use of the CSR 1000V is not limited to these use cases, and you can find many more with evolving functionality applicable in the future of computer networking. Further simplification of management will create more use cases for deployment of the CSR 1000V.

This page intentionally left blank

CSR Cloud Deployment Scenarios

This chapter describes the landscape you need to understand to fit the CSR 1000V amid your architecture. It discusses using CSR in a multitenant data center, with OpenStack, and in a public cloud environment.

CSR in a Multitenant Data Center

Data center designs have evolved, and the changes in technology have provided a multitude of options to design engineers. Today we commonly see network virtualization tied to security segmentation in the data centers. These network domains have localized services aligned to security segmentation. For example, it is common for enterprises to deploy virtual firewalls or load balancing services that are customized to their requirements inside the data center. This virtualization has evolved based on the availability of software and hardware features.

Chapter 1, “Introduction to Cloud,” describes virtualization in terms of server, network, and computing, as well as how things are brought together in the data center domain to create a multitenant data center environment. Virtual domains in a data center enable the data center to host multiple instances of virtual domains (that is, tenants) with the same physical hardware. Methods for virtualization can be hardware-based virtual logical units, like the virtual device concept of the Cisco Nexus 7000, or virtual routing instances in software. The multitenant data center infrastructure provides end-to-end logical separation for traffic flow and assets for multiple tenants. A multitenant data center is a cradle for IaaS that can be developed on this platform.

Multitenancy refers to logical separation of shared virtual computing, storage, and network resources. A tenant uses a logical separation—which can be a business unit, department, or workgroup—on the same physical hardware.

Figure 6-1 shows a logical overview of a multitenant data center.

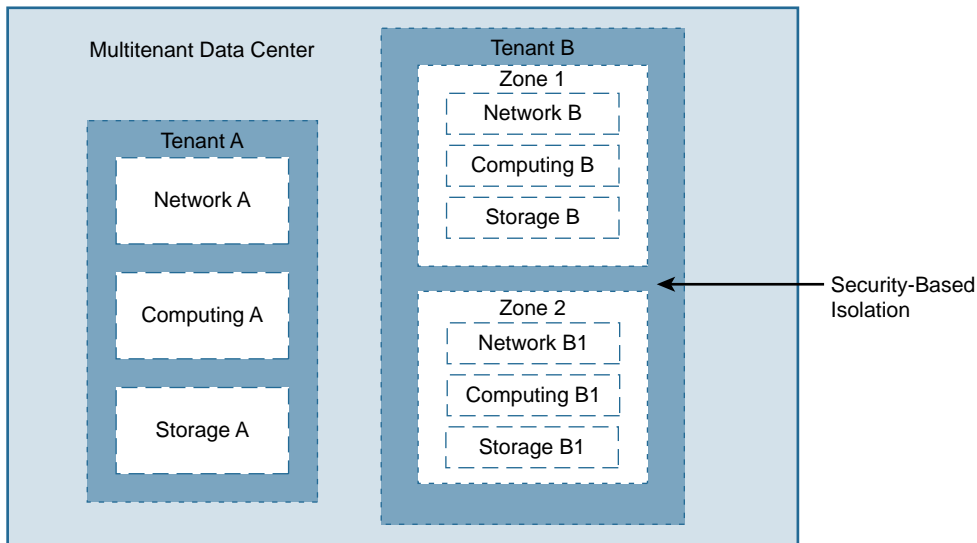


Figure 6-1 *Multitenant Data Center Logical Diagram*

Most data centers have tenant isolation and zones aligned to security policy to create further separation within the tenant infrastructure. These zones can be aligned for regulatory compliance of lines of business that require data privacy. In Figure 6-1, each zone in tenant B has separate network, computing, and storage infrastructures. However, in complex designs, zone resources in a single tenant domain can be shared. For example, zone 1 and 2 in tenant B could share the same storage domain.

The key design factor in a multitenant data center is the placement of the service block. This placement affects the characteristic of a virtual zone. Figure 6-2 shows a simple tenant zone with security and load balancing services aligned to it. Tenant A is a completely isolated, contained environment without services, whereas Tenant B and Tenant C have load balancing or a combination of load balancing and firewall services aligned with the contained zone environment.

A service block can be a physical or virtual device in a virtual zone. The segregation of traffic and isolation of the flow depend on the virtualization technology used in the zones. To have a simple traffic flow, it is recommended that the service block be placed near the server (asset). Placing the service block with the firewall close to the server helps simplify the plan to extend the domain across a single location as is normally done for disaster recovery. This simplifies the flow across the security infrastructure and prevents asymmetric flows that would be seen across firewall infrastructure.

Figure 6-3 shows microsegmentation within a single tenant, based on the policy of the security gateway.

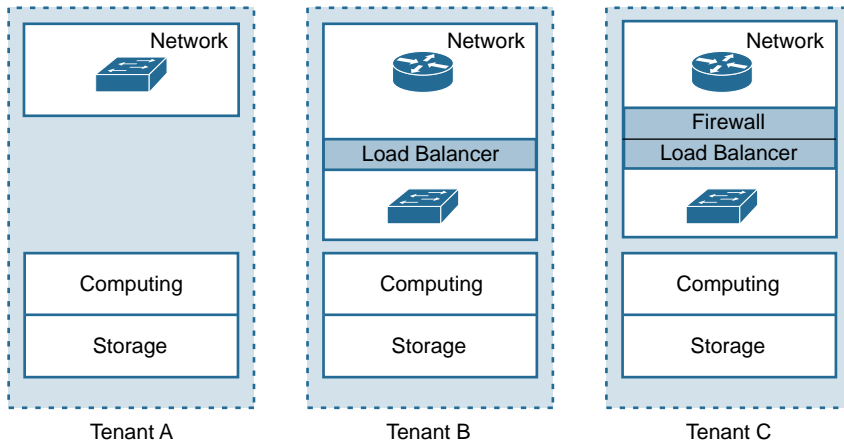


Figure 6-2 *Virtual Service Block Design in a Multitenant Data Center*

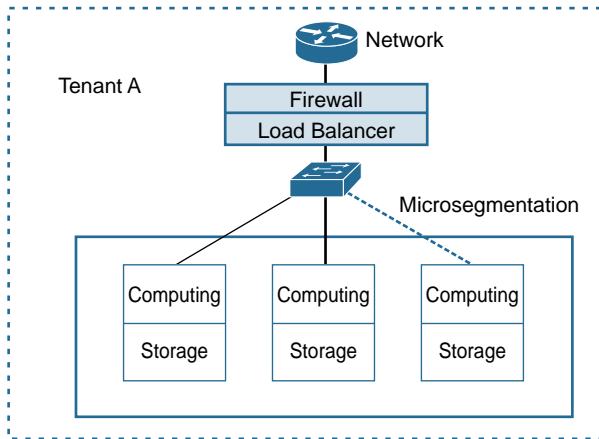


Figure 6-3 *Service Block Placement*

The following design methods can be used to achieve segmentation at the service level:

- A single context service block that aligns the traffic to various logical segments
- Virtual domains within the service block devices that are aligned to each logical segment

If the service block is deployed away from the computing and storage assets, network virtualization technologies such as virtual routing and forwarding (VRF) need to be used. Network segmentation can be done to carry the traffic with logical isolation to the computing and storage assets. With the rapid adoption of computing virtualization, the simplicity and flexibility of service block deployment is due to NFV elements such as firewalls, routers, or load balancers. The use of the CSR 1000V here allows an architect

to provide network elements (with extensive routing features, such as NAT and IPsec) and security elements (such as zone-based firewalls) in one virtual device instance. One of the main reasons for data center architects to consider physical service block devices is data throughput.

Figure 6-4 shows a classic traditional data center deployment with the service block (security gateway) creating multiple zones extended via VRF (Layer 3 segmentation) and VLAN (Layer 2 segmentation) technologies to create logical isolation.

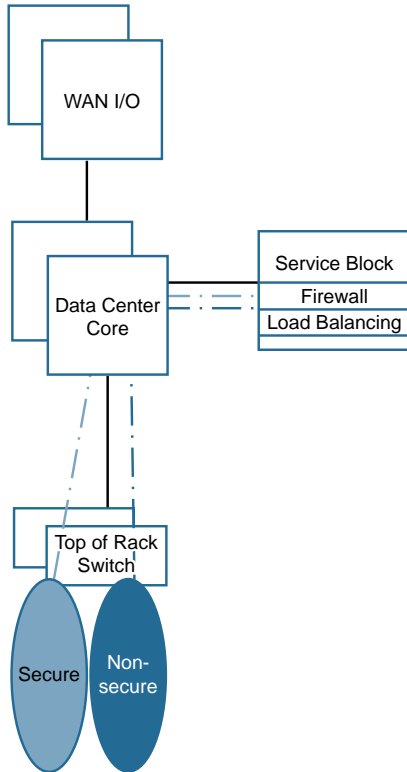


Figure 6-4 Classic Data Center Deployment with a Service Block

The CSR 1000V can be used to extend the security container to other data centers or cloud environments. The use of the CSR 1000V helps simplify traffic flow by creating a symmetric path across the security gateway segments. Features like overlay transport virtualization (OTV), VxLAN, and MPLS over GRE are used to extend the multitenant container across the WAN to another data center or cloud environment. This extension can be for Layer 2 or Layer 3 domains. Figure 6-5 shows connectivity of zones in a multitenant data center environment using the CSR 1000V.

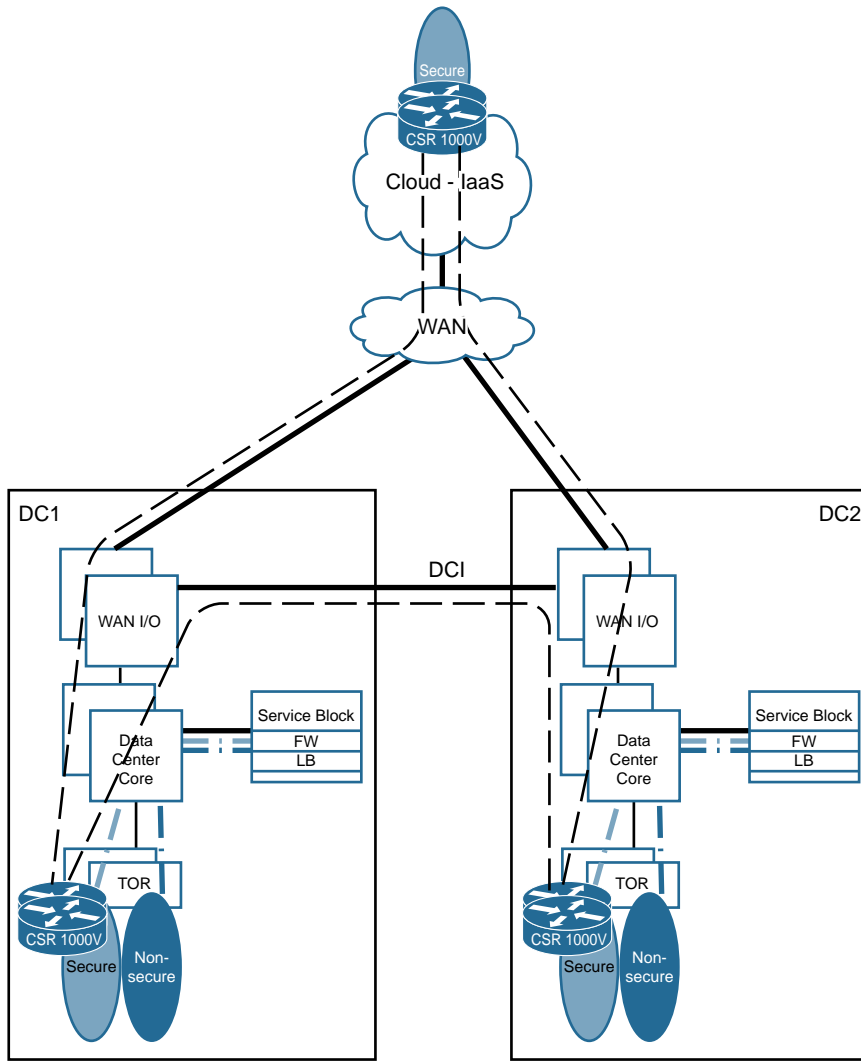


Figure 6-5 Connectivity of Zones in a Multitenant Data Center with CSR 1000V

In the example shown in Figure 6-5, the paths for the enterprise traffic to access the secure zones in the data center (DC1 and DC2) are via the local firewalls. The service block is localized within the data center. There are multiple options for maintaining the symmetric traffic flow to take care of the state inspection of the security gateway. The commonly deployed methods are NAT, selective routing, and firewall clustering. The different options for the systemic traffic flows between security gateways are not described in this book. The packet inside the secure tenant can access resources destined for this tenant. This tenant is extended between two data centers and a virtual data center (VDC) provisioned in the cloud. The traffic flow between the secure domains in each

of the data centers, through the physical firewall, needs to have policy set for the flow that increases the operational overhead on provisioning applications in the secure zone and increases complexity during troubleshooting. By adding NFV components for the CSR 1000V at these zones, you can isolate the traffic flow between these zones from the egress security gateway points. The CSR 1000V offers multiple overlay technologies, including the following:

- **OTV (Overlay Transport Virtualization) and VxLAN (Virtual Extensible LAN)**—Used for extension of Layer 2 domains across a DC or the cloud.
- **MPLS over GRE**—Used for extension of tenants' Layer 3 subdomains or Layer 2 extensions.
- **DMVPN overlays**—Commonly used for multipoint GRE with encryption.
- **LISP**—Provides overlays for location identification, device mobility, and carrying multitenant Layer 3 networks.

Apart from overlay, security gateway and VRF-Aware Software Infrastructure (VASI) functionalities, other technologies can be used to provide a multitude of services in the security domain. By using the CSR 1000V in the zone design, a data center architect can build preset logical communication between the zones within the data center owned by the enterprise and extend it to the cloud to leverage cost-effective provisioning within the boundaries of the security container defined and controlled by the enterprise firewall policy.

Cloudburst

The concept of leveraging the public cloud for deploying applications needing resources in spurts is called *cloudburst*. In this model, the application runs in a private cloud or data center and, based on demand, provisions assets into a public cloud environment. This is done to handle sudden spikes in the computing power. This model is also called the hybrid cloud environment, and an organization pays for extra computing resources only when they are needed.

Consider the following when planning for a hybrid cloud infrastructure that has cloudburst capability:

- Network connectivity:
 - Unified user access to the asset segment, regardless of where the asset is located (private or public cloud infrastructure)
 - Secure tenant space extension between the private and public infrastructures (the policy enforcement of both environments should be handled by a single management tool)
 - Management of the user experience via complete communication visibility

- Data synchronization:
 - The ability of an application to run and the stored data to be synchronized (based on application need) for a uniform user experience
- Workload migration:
 - The ability of the server and application-provisioning environment to leverage the same portal to host services in a private or public cloud (note that in a private cloud environment, you tend to use management tools for an enterprise server block)

The user access to the asset, whether hosted in a public or private cloud, needs to be uniform from a flow perspective, aligned to the application stack (web, middleware, and database deployment). This needs to be well planned. Two main models can be leveraged based on requirements for traffic flow: the direct access model and the redirection access model.

Direct Access Model

The direct access model defines IP transport connectivity from the user to the asset located in the private data center, or on-premise data center. This model is depicted in Figure 6-6. The cloudburst model is used for asset provisioning for computing resources on demand by the computing asset provisioning module. The traffic flows from the private data center to the enterprise asset in the public cloud.

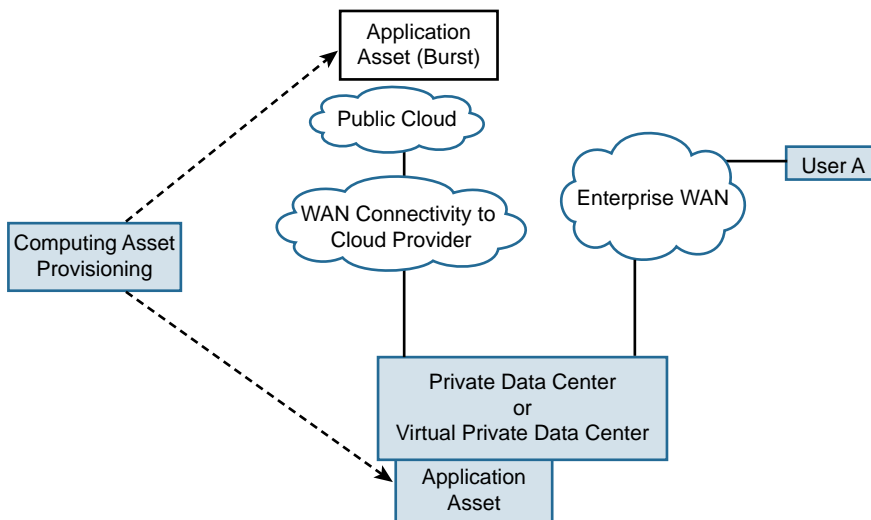


Figure 6-6 *Direct Access Model*

The enterprise connectivity to the public cloud is via the WAN connection from the service provider to the public cloud environment. In this example, the enterprise traffic access flows via the enterprise DC (or VDC for the tenant) to the WAN. In this case, the

cloudburst model considers only scaling of the computing resources. This model does not provide a seamless user experience for application access due to hair-pinning of traffic. However, this model does provide scaling of on-demand computing resources to the public cloud, and it thereby reduces the capital and operational costs for maintaining hardware resources to take care of the scaled demand. The tenant extension and isolation, if required, are covered by deployment of NFV components.

Redirection Access Model

The redirection access model is similar to the direct access model in terms of computing asset provisioning or scaling from a private data center to the public cloud, but it also provides a few extra capabilities to the enterprise, such as the following:

- Unified user experience for application access for data in the on-premise/private data center environment and to the public cloud
- Use of cloudburst for disaster recovery for active/active or active/standby

Using cloud-based on-demand or static assets for disaster recovery, especially for active/standby, gives the end user a cost-effective DR strategy that can be offered to multiple application tiers. Application stack design should be reviewed before cloud adoption. There might be limits to traditional applications that are not cloud ready to adapt to this model (see Figure 6-7), and this model applies only for cloud-ready applications that require locational availability.

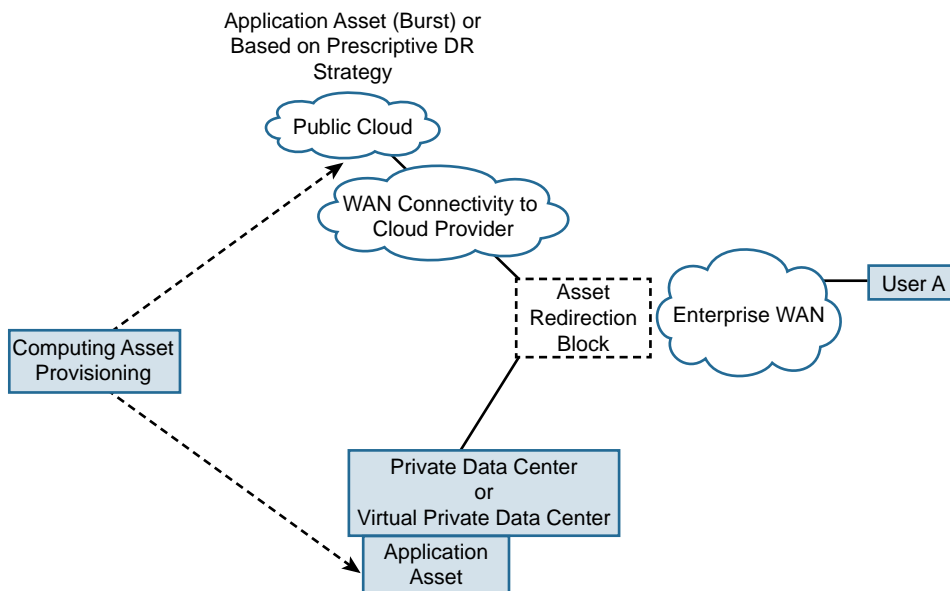


Figure 6-7 *Redirection Access Model*

Figure 6-7 shows the redirection access model. The asset redirection block is a key element that provides redirection of traffic to the private or public cloud. The asset redirection block is an architectural module that can reside in the private data center environment or at WAN hub locations at the enterprise data center. The use case of adding the asset redirection blocks at the enterprise WAN is for networks that are geographically spread around the world, with the logic abstracted to regional WAN hub locations. If the logic for redirection is based on DNS and location-based proximity, you can use hierarchical load balancer design. In this design, a global site selector redirects traffic to the local load balancer, based on different load balancing algorithms. The local load balancer can distribute traffic to local assets based on a defined algorithm taking care of the local load balancing criteria. The load balancer method is beneficial if you use application-centric probes to redirect traffic. Another way to achieve the distribution and intelligence from a network perspective is to use LISP. You have already read about the use case of LISP with the CSR 1000V in Chapter 5, “CSR 1000V Deployment Scenarios.” You can also use LISP for detecting asset mobility of a host from a private data center to a public data center. LISP is a more IP-centric approach for redirecting traffic based on asset location.

There are several use cases for the redirection access model, and two of them are described here:

- **Active/Active flow distribution to cloudburst**—In this scenario, the traffic from the user is directed to a private data center as the primary preference. The redirection can either be DNS based, using a hierarchical load balancer, or it can use network-based traffic redirection with LISP. The asset redirection logic is abstracted to the architectural block. In the event of an application demand increase, new computing resources are provisioned at the public cloud location, and the asset redirection block logic is changed to active/active flow. The user latency dependence is based on the access link types (see Chapter 1) from the private data center to the public cloud. By distributing this intelligence in the WAN aggregation block of enterprise customers, the user experience can be localized based on access connectivity of the region. The assets moved to the cloud would include the complete application stack for the user function. The database writes conditions between the locations, and usage in active/active model needs to be considered before selecting this method for active/active flow.
- **Disaster recovery using cloudburst**—In this scenario, the concept of cloudburst is used for asset provisioning during a failure scenario. This use case allows an enterprise to reduce the cost of its disaster recovery (DR) strategy. The asset redirection block directs traffic based on the application DR strategy. Unavailability of the asset at the primary location is determined by the probe (generated by a network or hierarchical load balancer), which redirects the traffic to the new location. The provisioning of the asset in an active standby scenario depends on the computing asset provisioning portal.

The Cisco Inter-Cloud Fabric

The Cisco inter-cloud fabric framework handles cloudburst use cases. This fabric takes care of separate individual functionalities, such as extension of tenant domains, API mapping from private to public cloud, traffic control, and ability to dynamically spawn network services and create user-defined traffic flow within a single management view. Inter-cloud fabric provides the following benefits:

- Provides on-demand provisioning of services
- Is simple to manage and deploy
- Creates automated extension of Layer 2 domains that helps in preserving the IP space aligned to the data center
- Allows for a single management portal for all the technical domains
- Offers easy deployment of security and network features

Figure 6-8 shows the important elements of the inter-cloud fabric framework. This section describes the functionality of these elements, which contribute to the simplification of the inter-cloud solution and enable enterprises to cloudburst.

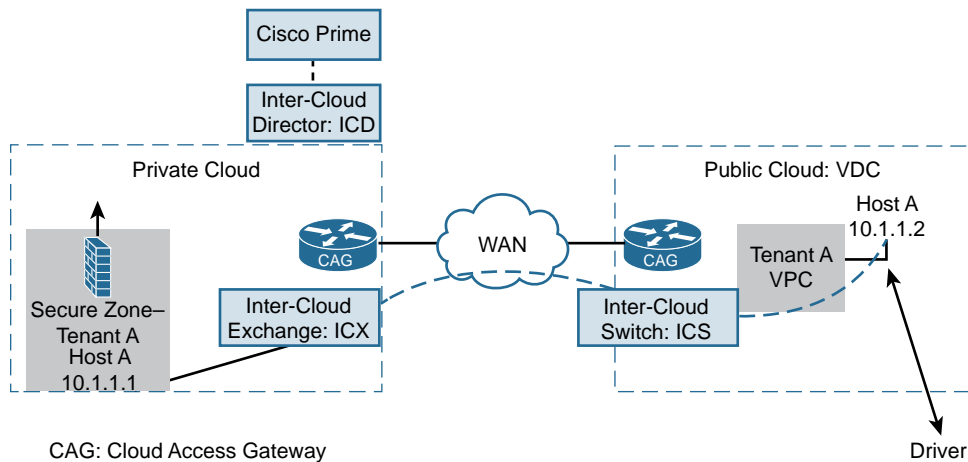


Figure 6-8 *Inter-Cloud Fabric*

Figure 6-8 shows the components of the inter-cloud fabric. The inter-cloud director is an OVA file that launches a VM that is used for management and provides a single operational view of the inter-cloud solution. This file is also used for API mapping between different cloud providers or private clouds by creating a template. The template is a user-defined profile for a tenant asset that pulls all APIs into a user-defined grouping. This grouping has a parent and a child profile. The parent profile is the cloud (public or

private) API index, and the child template is the new inter-cloud template for the tenant that pulls in all the attributes into one single template.

The inter-cloud exchange (ICX) is an inter-cloud switch (ICS) extension of the tenant transport segment between the private and public clouds on a carrier transport between the cloud domains that has already been provisioned. The inter-cloud exchange is positioned on the private cloud that connects to multiple public cloud providers. At the public cloud, the ICS is provisioned. The data plane for the tenant extension from the private cloud to the public cloud data flows between the ICX and ICS, across an encrypted path that has the capability to carry Layer 2 domains. The encrypted domain is maintained for the host of the driver at the public cloud. The host driver in the Cisco inter-cloud framework at the remote end encrypts and decrypts the data plane. The communication between the hosts at the public cloud is contained within the tenant domains maintained at the private cloud. Each tenant domain has a separate encryption key that isolates the communication in the public cloud. The communication needs to traverse through the ICS.

The ability to host virtual services for networking and security is possible from the inter-cloud fabric director, which can instantiate the CSR 1000V or virtual security domain appliance at the public cloud. The ability to add the CSR 1000V on demand enables a data center admin to take care of different use cases, where the traffic flow might need to use the public cloud exit point and where NAT will be necessary for the enterprise IP address to preserve and use the cloud provider as an exit point. The NAT policy masks the enterprise IP address to the public cloud's IP address.

Host A (10.1.1.1) in Figure 6-8 at the private data center needs to communicate with the 10.1.1.2 host ported in the public cloud. The traffic pattern needs to follow the same security rule set. The secure tenant zone has a network connection to the cloud access router; the inter-cloud exchange device builds a Layer 2 encrypted tunnel to the inter-cloud switch hosted in the public cloud. If the 10.1.1.2 computing asset has not been provisioned, it is possible to use the inter-cloud director to spawn a new VM from a single portal. The communication between 10.1.1.1 and 10.1.1.2 is encrypted. In the future for the same enterprise, if another VM is provisioned outside the secured container at the public cloud—say 10.1.2.1—the communication between 10.1.1.2 and 10.1.2.1 is not possible because they are in two separate security zones extended to the public cloud. Both the hosts will have encrypted communication with the ICS. Because they exist in two separate security domains, the encryption key will be different and will be considered mutually exclusive.

Private Cloud Deployment with CSR in OpenStack

The following sections provide an overview of OpenStack and how the different OpenStack components come together to create a private cloud.

Introduction to OpenStack

With open source software gaining momentum, customers want software that is free of cost and open for changes so that they can tailor the software to suit their needs. OpenStack is a cloud operating system that is a free and open source cloud computing platform. Written in Python, OpenStack consists of different software components that were developed as separate projects but come together to work as a single cloud operating system designed to manage a cloud environment.

OpenStack is managed by a nonprofit entity called the OpenStack Foundation, which has 500 member companies. OpenStack is released under the terms of the Apache license and has a release every six months.

OpenStack enables the end user to automate and manage infrastructure. It has become very popular for private clouds because it enables the user to maintain its own IaaS.

Primary Use Case for OpenStack

OpenStack, as a cloud operating system, was primarily designed to address the computing needs of groups of users. It is therefore best suited for automating an IT infrastructure that caters to the needs of an organization or a group of users.

Linux is a very powerful operating system, and the KVM module in Linux enables it to virtualize. As discussed in Chapter 3, “Hypervisor Considerations for the CSR,” Linux (along with KVM and QEMU) is a fully functional type 1 hypervisor. It enables you to manage your hardware infrastructure, spawn VMs, and automate. So if you already have Linux, why do you need OpenStack as an operating system? As mentioned earlier in this chapter, it’s a cloud operating system, which makes it very distinct from regular operating systems like Linux.

Linux, along with KVM and QEMU, gives you a fully functional hypervisor and allows you to create virtual machines. However, Linux can do this only for the single server blade that it runs on. If you have multiple server blades and you want to pool in the memory, CPU, storage, and network resources of all of them, you cannot use a single instance of Linux to do that. Even if you run Linux on all servers, you still do not have a way to get the resources of all servers together and manage them as one entity. OpenStack solves this problem. It gives you one holistic view of all the resources at your disposal. It pools the hardware resources available and manages them for you, as an operating system would. Since OpenStack has the ability to do this for a bunch of resources and you can replace hardware at will without impacting the functioning (and thus scale), it is rightly called a cloud operating system.

Figure 6-9 compares the capabilities of Linux and OpenStack. You can see here that Linux is used to manage a single server, while OpenStack is used to manage pooled computing, network, storage, and memory resources.

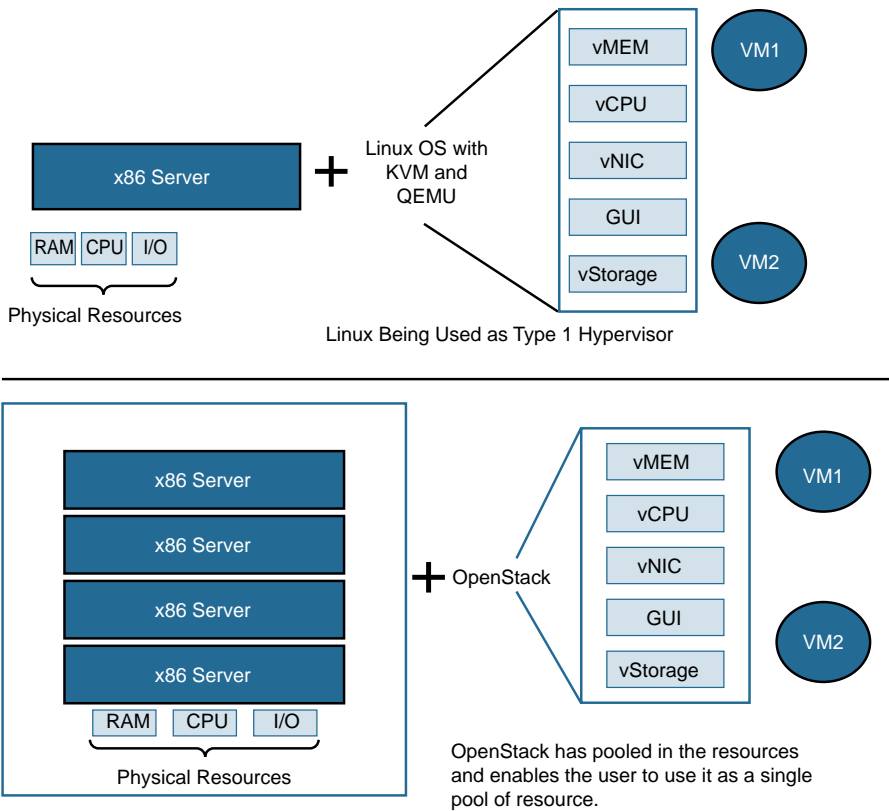


Figure 6-9 *Linux Versus OpenStack*

OpenStack Components

OpenStack is an operating system that enables you to create, automate, and monitor your cloud. OpenStack is made up of individual components written in Python. Each component, developed as an individual project, performs a specific role in the OpenStack framework, as shown in Figures 6-10 and 6-11.

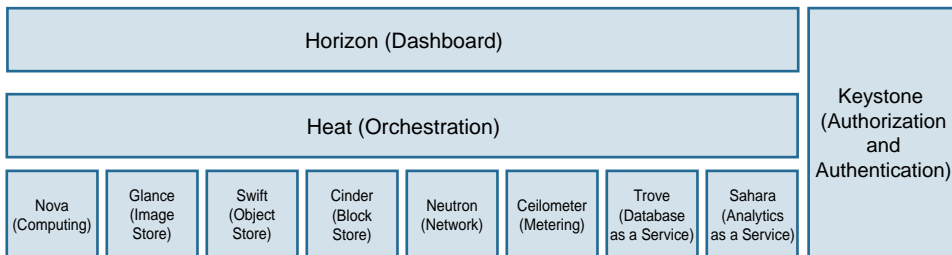


Figure 6-10 *OpenStack Projects*

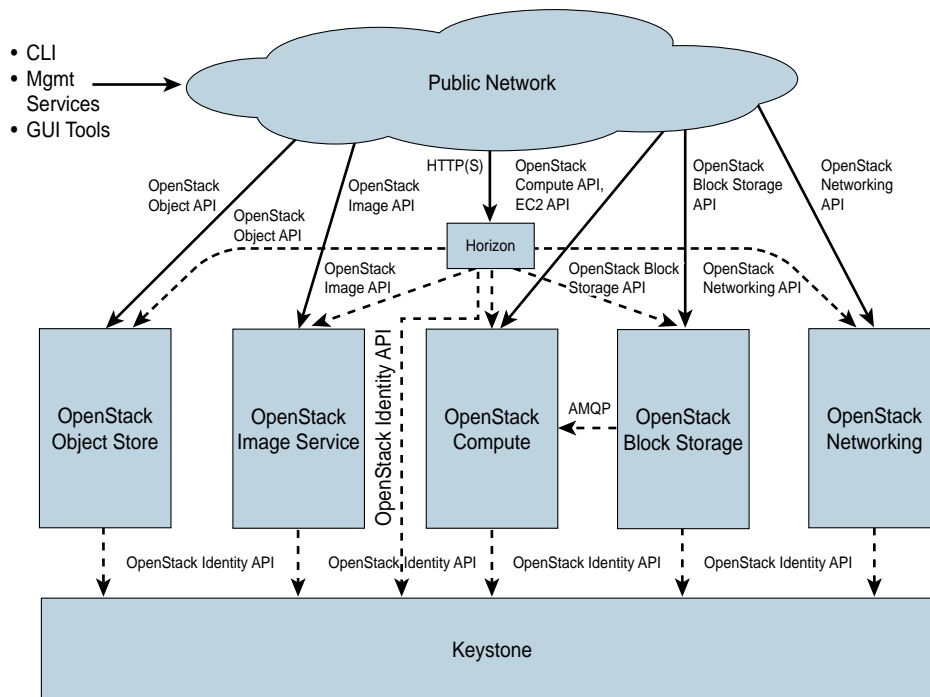


Figure 6-11 *OpenStack Architecture; Each Project Is Accessible via a Project API, and Users Can Access OpenStack via the Dashboard (Horizon) or Through Project-Specific APIs*

Nova (Computing)

Nova sits at the heart of the OpenStack framework. It is the computing engine that is responsible for creating and maintaining virtual machines. It is a complex and distributed component of OpenStack. Multiple processes constitute the Nova computing project. Here is a brief description of some of the processes in the Nova project:

- `nova-api`—This process accepts and responds to the end user’s computing API calls. It is responsible for orchestrating events like running a virtual machine. It is also responsible for enforcing policy.
- `nova-compute`—This process creates and terminates virtual machines. It uses hypervisor calls to create and terminate VM instances:
 - With KVM/QEMU, it uses `libvirt`
 - With XenServer/XLP, it uses `XenAPI`
 - With VMware, it uses the `VMware API`

Figure 6-12 shows logically how this is achieved.

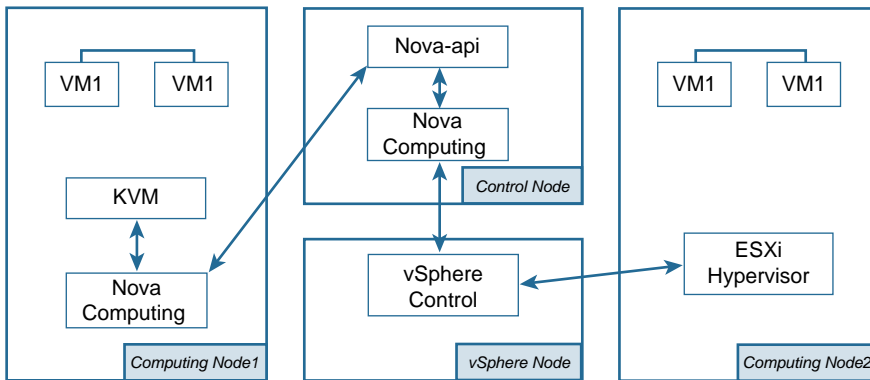


Figure 6-12 Hypervisor Interaction

- `nova-scheduler`—This process is the scheduler for the Nova computing engine. It determines which host is best suited to spawn the next VM request.
- `nova-network`—This process picks up a task from the queue and executes the task in a way similar to `nova-compute`. It targets the networking piece of OpenStack, like provisioning Layer 2 and Layer 3 circuits, setting up IP addresses, and handling NAT and other network functionalities. `nova-network` functionality moved on into a separate project, called Neutron. Neutron is a full software-defined networking stack. However, `nova-network` is still used for simple use cases where the networking requirements are limited.
- `nova-xvncproxy`—This process daemon is a proxy used to access the virtual machine instances through a VNC connection.
- **The queue**—The message queue connects daemon processes and serves as a central repository for passing messages between processes. It is implemented using Advanced Message Queuing Protocol (AMQP), a standard message queuing protocol for passing messages between applications.

Note There are multiple other processes in the Nova project. Covering all of them is beyond the scope of this book. OpenStack projects are well documented online at <http://docs.openstack.org>.

Keystone (Identity Service)

Keystone provides authentication and authorization services in OpenStack. It is the project that's responsible for setting access rights and making sure only authorized users are able to access the system. It also guards access rights between projects in OpenStack. All services in OpenStack rely on Keystone to verify requests. Keystone takes care of API

requests and provides users with a catalog of available offerings, enforces policy, and administers token and identity services.

Keystone issues a token, which essentially defines the scope of access in OpenStack. The token issued is based on the user's credentials. Figure 6-13 illustrates how keystone works at a very high level. These are the steps shown in the figure:

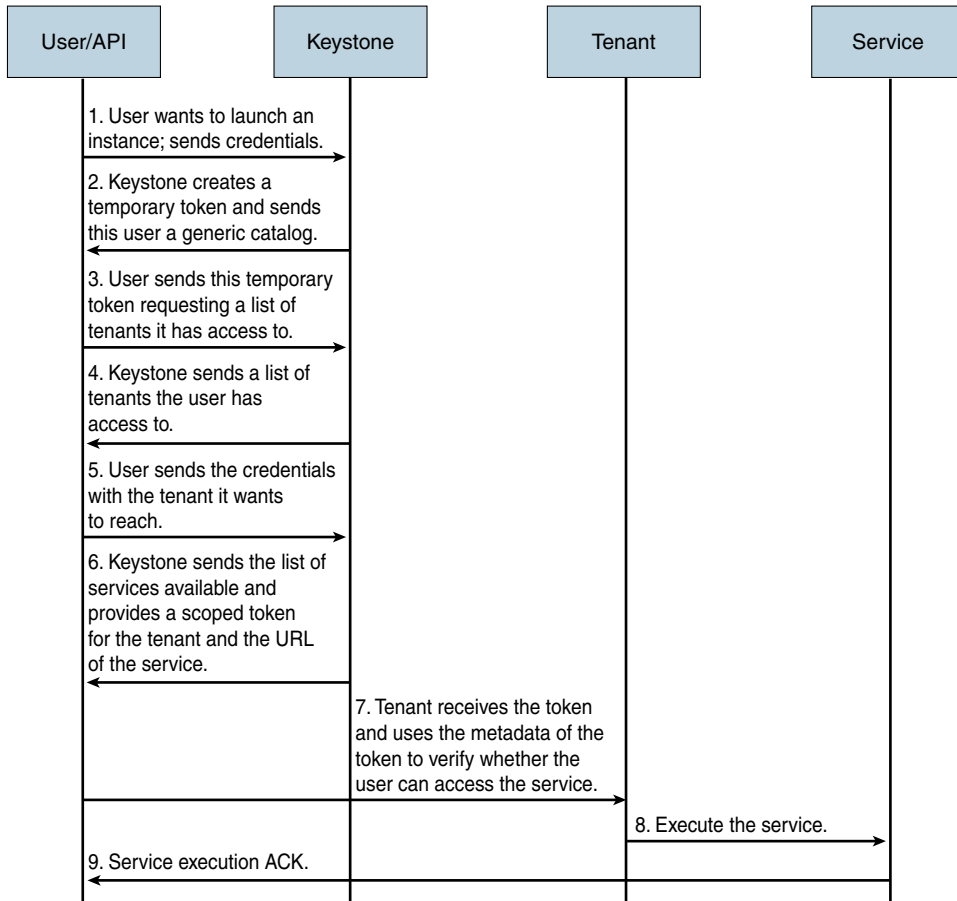


Figure 6-13 *Keystone Function Overview*

Step 1. The user sends an unscoped token (that is, a token that is not tied to any particular project in OpenStack) to determine which OpenStack projects/tenants the user has access to. This token is strictly used to query Keystone to determine which projects the user has access to. It is important to note that only scoped tokens (tokens that have user credentials and the tenant details where the service is requested) must be used with non-Keystone projects, such as Nova. Unscoped tokens are only used to query Keystone. The user provides the credentials (username and password).

- Step 2.** Keystone returns an unscoped token to the user.
- Step 3.** The user sends the unscoped token to determine which tenants/projects he has access to. This step typically uses a REST API `get` with this unscoped token and sends it to Keystone.
- Step 4.** Keystone responds with a list of tenants to which the user has access.
- Step 5.** The user now has access to a list of tenants he can access and must decide which tenant he wants to use. To work with any tenant, the user must obtain a scoped token. A scoped token is always for a particular tenant and contains metadata for the tenant. To obtain a scoped token, the user now does a REST API `post` with username and password, such as in Step 1. The difference here is the tenant/project name. In this `post`, the user specifies his username and password that align with the tenant and project name for which the token is requested.
- Step 6.** As a response to the `post`, Keystone sends a scoped token with the metadata associated with the tenant/project. The user now has a service catalog that contains the URLs to the tenant/project.
- Step 7.** The user can now invoke the service by using the scoped token and the target URL of the service he wants to invoke. (This is the case with UUID-based tokens only.) The tenant receives the service request with the token. It uses the metadata of the token to verify the user access rights for the service. The `policy.json` file for each tenant determines the role-based access.
- Step 8.** The service request is executed.
- Step 9.** The user receives an API response.

Keystone is the first project to be installed during OpenStack deployment.

Note It's important that you understand how to relate multitenancy concepts with OpenStack:

- *Tenants* are represented as projects, and a *project* is the base unit of “ownership” in OpenStack. A tenant can be defined as a user assigned a fixed set of resources within a container. This is an important concept in multitenancy for network, storage, and computing domains.
- A *group* is a collection of users that are part of a domain.
- A *domain* is a collection of projects and users.

Glance (Image Service)

Glance is an image service project in OpenStack that stores and discovers data that is meant to be used by other services in OpenStack. It discovers, registers, and downloads virtual machine images. In other words, it is the local image repository in OpenStack. Glance also stores metadata information about the images it stores.

Glance uses a client/server model. The applications that use Glance are clients, and the Glance project behaves as a server. It uses REST API as the communication mechanism. Following are the major components of Glance:

- **The client**—This is the application that uses Glance.
- **REST API**—Glance is externally accessible via REST API.
- **DAL (Database Abstraction Layer)**—This is an API that sits between Glance components and the Glance database.
- **Domain controller**—The Glance domain controller is responsible for authorization, notifications, policies, and database connections.
- **Glance store**—This is the middleware between Glance and various data stores that are supported by Glance.

Figure 6-14 illustrates the architecture of the image service project, Glance, in OpenStack.

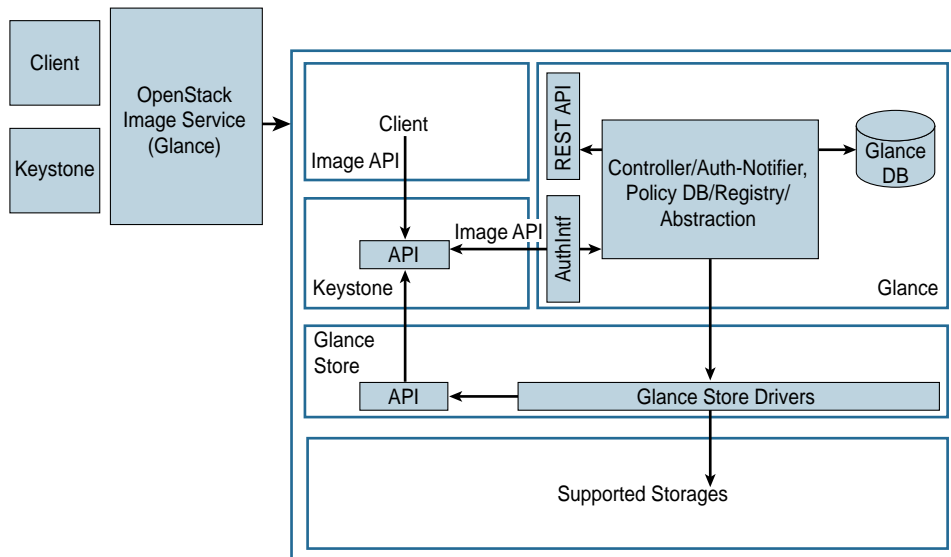


Figure 6-14 *OpenStack Glance Architecture*

Neutron

Before Neutron became a separate project, it was Nova's networking service, `nova-network`. You can still use `nova-network` for basic networking operations. It can do base Layer 2 network provisioning, IP address management for tenants, DHCP, DNS, and enable implementation of firewalls and NAT using IPTables.

Neutron was created as a separate project from Nova to address the limitations of `nova-network` and bolster the networking capability of OpenStack. `nova-network` had multiple limitations; for example, it did not have APIs for consuming networking services, and it had a VLAN-based network model, which allows only 4096 VLANs and only a very limited set of features. Perhaps the most striking drawback was that the architecture did not allow plugins to render their functionality to OpenStack. Neutron, which was promoted to a core project at the Folsom Summit in April 2012, addresses the following functionalities:

- **Modular Layer 2 architecture**—Before this architecture, each new Layer 2 service hogged the entire Neutron server process. Each time you came up with a new Layer 2 plugin, the entire Neutron server was dedicated to running that plugin. This was not only a problem from an operational perspective but was an issue with developing the plugin, too, because a lot of code needed to be rewritten.

With the modular Layer 2 architecture, vendors can seamlessly integrate plugins to Neutron and use the different Layer 2 encapsulations that these plugins bring in. For example, with this architecture, you can have one tenant use the dot1q VLAN while another uses a VxLAN.

Figure 6-15 illustrates the Modular Layer 2 (ML2) architecture of Neutron.

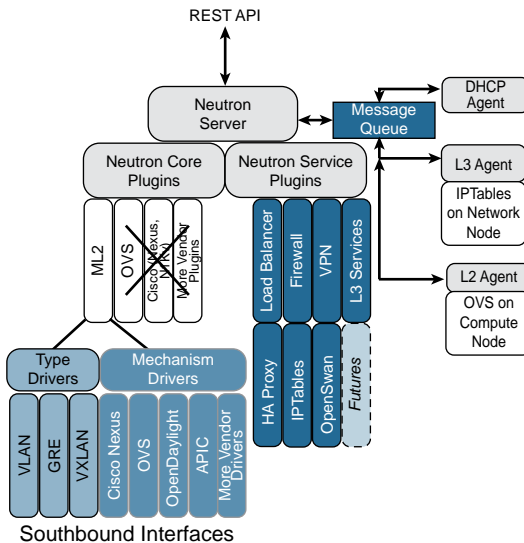


Figure 6-15 *ML2 Architecture: Neutron*

- **Tenant APIs**—Tenants can use API calls to create as many networks as they want. Whenever you bring up a VM, you just need to define the networks it should connect to.
- **Features**—`nova-network` was missing features like advanced ACLs and QoS. Neutron lets you get all these features into OpenStack.

Cinder (Block Storage)

The OpenStack wiki says Cinder “virtualizes pools of block storage devices and provides end users with a self service API to request and consume those resources without requiring any knowledge of where their storage is actually deployed or on what type of device.” Fundamentally, Cinder provides persistent storage to guest virtual machines of OpenStack.

Block storage is one of the most fundamental requirements for a virtualized infrastructure. This is because the virtual infrastructure is maintained as files. Each virtual machine is essentially a file that needs storage. These VM files and the data associated with them need to have persistent storage so that the VM can be powered down and brought up to life whenever required.

Cinder is the block storage mechanism within OpenStack. It stores and provides access to block storage for OpenStack tenants. To OpenStack tenants, this storage appears as a block device that can be accessed using mechanisms like iSCSI, Fibre Channel, and NFS for back-end connectivity.

Figure 6-16 illustrates the architecture of Cinder, which has the following components:

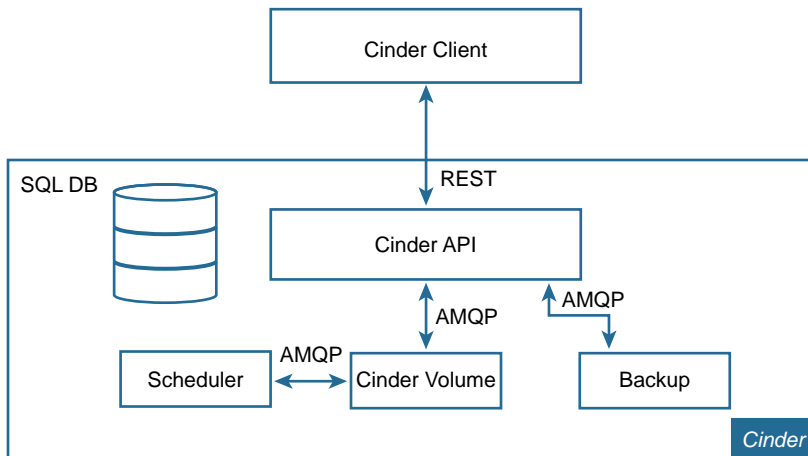


Figure 6-16 *Cinder Architecture*

- **Cinder API**—This piece of code, as with all other software in OpenStack, talks AMQP with modules inside the project. With modules outside the project, it talks REST API. Cinder API takes in REST API requests from the north and routes them to the appropriate Cinder modules talking AMQP.
- **Cinder scheduler**—This component schedules the requests for the volume service.
- **Cinder volume**—This is the block storage manager that manages back-end storage devices.
- **Cinder backup**—This component backs up Cinder volumes.

Other OpenStack Projects

The following are some of the other projects in OpenStack:

- **Horizon (dashboard)**—This is OpenStack’s dashboard. It is a web-based self-service portal that the user can log in to after installing OpenStack. Architecturally, it sits above all OpenStack projects and interacts with all projects using APIs. It gives the user some very useful functionalities, such as instance creation, network configuration, and storage. This interface can also be used to complete some administrative tasks, such as user creation.
- **Heat (orchestration)**—This OpenStack project, like Horizon, interacts with all components of OpenStack. It is used to create application stacks from multiple resources (like servers, floating IPs, security groups, users, etc.). It is an engine that is used to launch multiple cloud applications based on templates. A Heat template describes the skeleton for an application in a text file that can be edited.
- **Swift (object storage)**—This project provides a mechanism for storing and retrieving unstructured data. Swift makes sure that data is replicated across a cluster of servers and that the integrity is maintained. It provides distributed object storage. Storage clusters scale horizontally, and if one of the servers fails, OpenStack replicates the data from other active nodes to new cluster locations. Swift stores objects on object servers. Swift is implemented using a ring concept, where a ring is a component that contributes to the Swift service.
- **Ceilometer**—This project is used to collect measurements within OpenStack. OpenStack needs metering and monitoring services, and Ceilometer is responsible for giving OpenStack that service. It accesses and inserts metering data using REST API. Ceilometer has three kinds of defined meters:
 - **Cumulative meter**—This meters things that increment over a period of time.
 - **Delta meter**—This meters things that change over a period of time.
 - **Gauge meter**—This meters discrete (for example, images) and fluctuating (for example, disk I/O) values.
- **Trove**—This is a database as a service within OpenStack and is one of the newest projects of OpenStack. It aims to provide a scalable and reliable cloud database service. It enables you to create a database instance with the data store of your choice (for example, MySQL), as shown here:

```
$trove create -size <size of the DB> -users <username>:<PassName> -datastore
  MySQL <instanceName>
```

This example creates an isolated database environment with computing and storage resources.

- **Sahara**—Sahara provides a scalable data processing stack with a management interface. It achieves this by giving the client the ability to quickly create and manage Apache Hadoop clusters and run workloads across them.

Note OpenStack can manage different types of hypervisors and can also manage containers. What is container-based virtualization?

Container-based virtualization takes place at the application layer, in the operating system. Each application node is provisioned as guest virtual machines running on the same kernel. The kernel takes care of the hardware calls through a unified approach for all guest VMs. All the guest VMs need to run the same operating system, and this can be a limiting factor. The use of a single operating system can be a delimiting factor for use in the enterprise application space. However, in development or hosted service environments, using containers to run isolated guest VMs is a faster and simpler method than using traditional hypervisor modes.

CSR Within OpenStack

Neutron, or “OpenStack networking,” started off as a very basic networking requirement. The CSR, being a software router, has a lot to offer OpenStack networking in terms of features. OpenStack’s modular architecture allows services to be incorporated into it. CSR thus fits in perfectly. You can classify the services CSR can render to OpenStack into two broad categories, CSR 1000V as a Neutron router and CSR 1000V as a tenant router, which are described further in the following sections.

CSR 1000V as a Neutron Router

CSR brings feature richness to OpenStack. The role of a Neutron router in OpenStack is to network the virtual machines and make them accessible based on user requirements. CSR can be used to deliver Neutron’s Layer 3 routing service API. When working within OpenStack, CSR is managed by Nova. The entire life cycle of the CSR virtual machine is managed by a Layer 3 service plugin using Nova. This service plugin configures the CSR virtual machine to make sure the Neutron service is available through APIs using this service VM.

The CSR’s functionality is a superset of what the Neutron actually requires, so the CSR “hosts” the Neutron functionality within itself. A special admin tenant owns the CSR, and the OpenStack tenants do not have visibility to this. Tenants use the Neutron functionality exactly the way they do without the CSR. Internally, in the realm of this special admin tenant, the CSR hosts the Neutron functionality using two interfaces. One is a management virtual interface that is used to connect to the management network in OpenStack. The other is the tenant traffic virtual interface that trunks the internal tenant network and the external Neutron network.

Figure 6-17 illustrates how the Neutron services can be hosted by the CSR inside OpenStack.

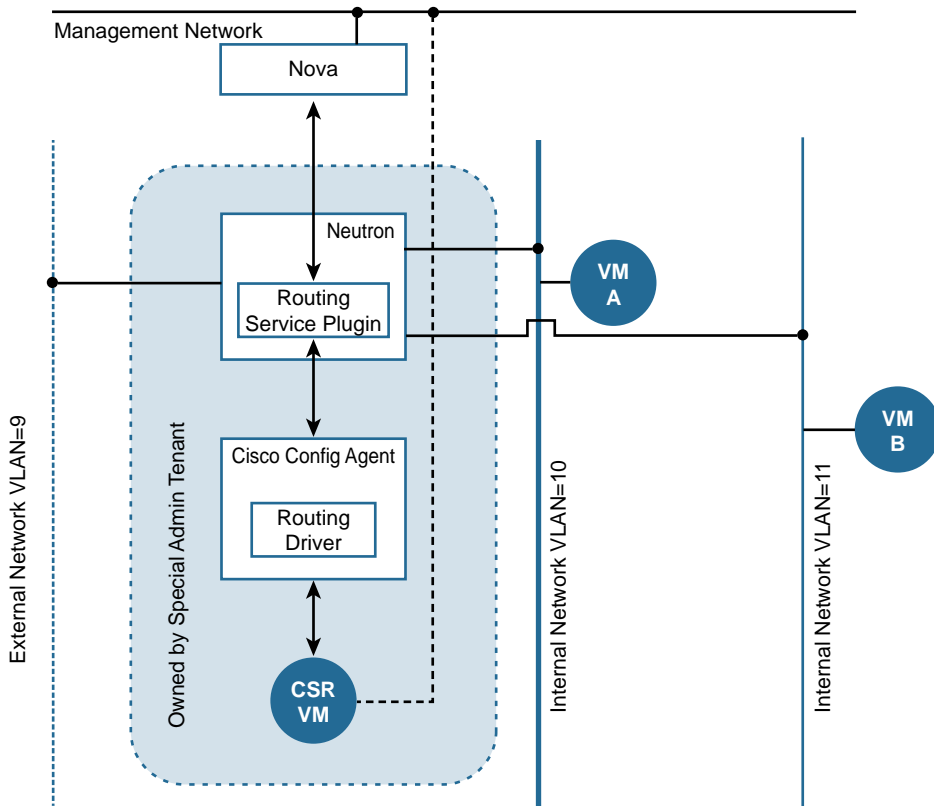


Figure 6-17 CSR Hosting a Neutron Router

OpenStack is essentially designed to talk to Neutron, and the CSR is a software router that needs to be created and managed, just like any virtual machine. To create a Neutron router hosted within a CSR VM, you need to install the following:

- You need to install a plugin in Neutron to talk to the CSR virtual machine and also to Nova. This spins up a CSR VM when instructed to do so by Nova.
- You need to install a configuration agent that talks CLI to the CSR and talks RPC (implemented using RabbitMQ) to the Neutron module. The routing service plugin in Neutron will discover the configuration agent (over the management network). The CSR virtual machines and the configuration agent communicate over the management network, which may or may not be the same as the OpenStack management network. In Figure 6-17, both management networks are the same.

Follow these steps to install CSR as a neutron router:

- Step 1.** To establish connectivity to Keystone, create the admin tenant that houses the CSR:

```
$ keystone tenant-create --name L3AdminTenant --description "Special Admin Tenant"
```

The name defaults to `L3AdminTenant`. If you want to change this, modify `/etc/neutron/cisco_router_plugin.ini`.

The command returns a UUID of the tenant created. Call it `UUID_Special_Admin_Tenant`.

Change roles to admin for the special admin tenant created above:

```
$ keystone user-role-add --user-id <UUID_Neutron_Service> --role-id
  <UUID_admin_role> --tenant-id <UUID_Special_Admin_Tenant>
```

Use the following Keystone commands to get the Neutron service UUID and admin role UUID:

```
UUID_Neutron_Service == keystone user-get neutron
UUID_admin_role == keystone role-get admin
```

Give the Neutron service admin access in the service tenant:

```
$ keystone user-role-add --user-id < UUID_Neutron_Service> --role-id
  < UUID_admin_role> --tenant-id <UUID_service_tenant>
```

Use the following Keystone commands to get the neutron service UUID and admin role UUID:

```
UUID_service_tenant == keystone tenant-get service
```

Step 2. Set up Nova and Glance for the CSR VM:

```
$ nova flavor-create csr1kv_router 621 8192 0 4 --is-public False

$ nova quota-update --cores -1 --instances -1 --ram -1 <UUID_Special_
  Admin_Tenant>

$ glance image-create --name csrImage --owner <UUID_Special_Admin_
  Tenant> --disk-format qcow2 --container-format bare --file <csrImage
  path> --property hw_vif_model=virtio --property hw_disk_bus=ide
  --property hw_cdrom_bus=ide
```

Step 3. To get Neutron to talk to the CSR routing service plugin, modify the Neutron configuration file and make sure you point the `service_plugin` variable to the CSR:

```
service_plugins=networking_cisco.plugins.cisco.service_plugins.cisco_
  router_plugin.CiscoRouterPlugin
```

Create Nexus 1000V (N1kv) network profiles. Execute the following commands as user `neutron` in the special admin tenant:

```
$ neutron cisco-network-profile-create --tenant-id <UUID_Special_
  Admin_Tenant> --physical_network osn_phy_mgmt_network --segment_range
  <vlanIdMgmt>-<vlanIdMgmt> osn_mgmt_np vlan

$ neutron cisco-network-profile-create --tenant-id <UUID_Special_Admin_
  Tenant> --physical_network osn_phy_network --sub_type VLAN osn_
  tenant1_np trunk
```

Step 4. Connect to the Cisco configuration agent by logging in as user Neutron in the special admin tenant. Then create a port for the configuration agent:

```
$ neutron port-create --name ciscoCfgAgent --tenant-id <UUID_Special_
  Admin_Tenant> --fixed-ip ip_address=<ipAddressCfgAgent> osn_mgmt_nw
  --nlkv:profile_id <mgmtUUID>
```

Now you need to log in to the server running the config agent and configure `ipAddressCfgAgent` on the interface and `macAddress` of the port you just created, using the previously shown CLI commands.

Step 5. To verify that the installation was successful, create external and internal networks:

- Create a Neutron router and attach the internal and external networks to it. When you do this, the CSR you set up in Steps 1–4 is created inside the special admin tenant and should work as a Neutron router.
- Attach interfaces, remove them, and delete them just as you would for a Neutron router:

```
$ neutron net-create public --router:external True
$ neutron subnet-create public <ipAddressPublic>
$ neutron net-create private
$ neutron subnet-create private <ipAddressPvt>
$ neutron router-create routerCSR
$ neutron router-gateway-set <routerCSR ID> <publicNetId>
$ neutron router-interface-add <routerCSR ID> <pvtNetId>
```

CSR 1000V as a Tenant Router

You can bring up a CSR 1000V as a tenant VM and network your tenants within OpenStack. The CSR 1000V will be spawned inside OpenStack, like any tenant virtual machine. This way, you can use the features that come with the CSR 1000V for the virtual machines attached to the CSR on the same network.

Note If you attach the CSR as a tenant router to the provider/public network, you do not need a Neutron router to NAT your traffic outside OpenStack. It is a practice to leverage a plugin when the CSR attaches directly to the provider network to replace a Neutron router. You will learn more about the CSR plugin in Chapter 7, “CSR in the SDN Framework.”

In Figure 6-18, a CSR is spawned as a tenant VM in each of the internal networks: 1 and 2. In each network, the CSR is being used as a firewall or VPN head-end. VM A and VM B use CSR Network 1 and CSR Network 2, respectively. This model gives you tremendous control over the firewall and VPN policies to impose on your networks.

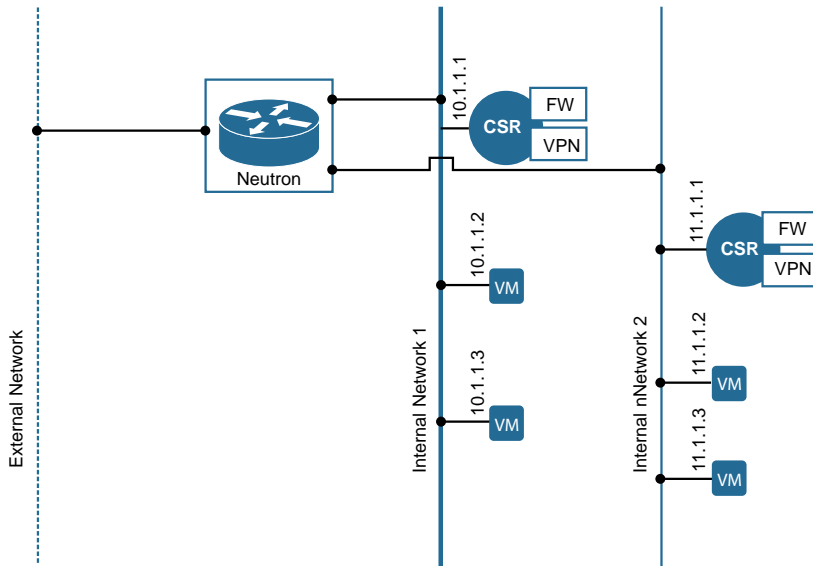


Figure 6-18 *CSR as a Tenant VM*

Figure 6-19 illustrates how to use the CSR to segregate a single network into multiple tenant networks.

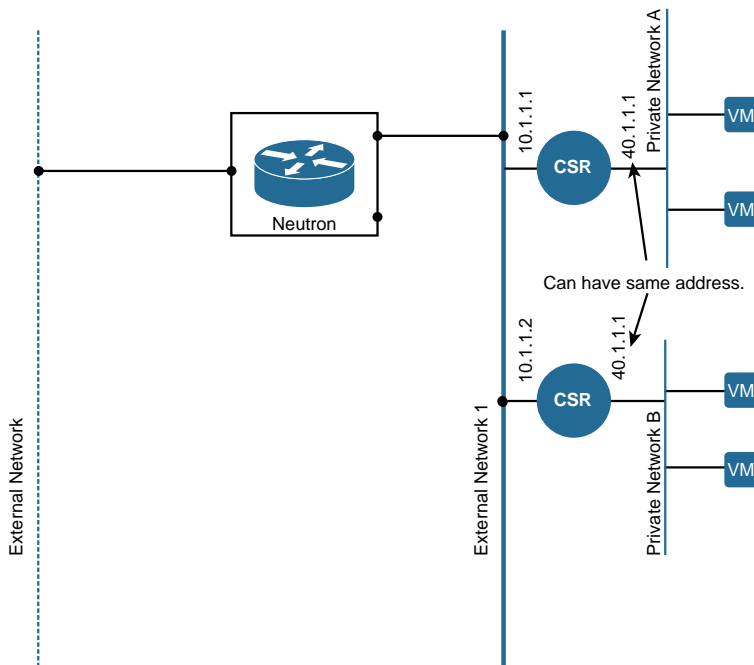


Figure 6-19 *CSR Segregating One Network into Multiple Tenant Networks*

To install the CSR as a tenant VM, bring up CSR just like any other tenant VM within OpenStack. Then just list the flavor and image list, choose one, and use `nova boot` to create it. Nova uses Neutron to get an IP address for the CSR you spawned. You can then configure the CSR using the console access and set up the required service for the other tenant VMs:

```
$ nova flavor-list
$ nova image-list
$ nova boot --flavor flavorType --key_name keypairName --image ID
newCSRInstanceName
```

CSR 1000V in a Public Cloud

The following sections cover CSR deployments in public clouds. The focus here is on AWS (Amazon Web Services).

Amazon Web Services Deployment for the CSR

You have already read about deployment of the CSR 1000V in a public cloud environment. This section will help you understand the details and considerations for such a deployment. The use case chosen to depict the public cloud is AWS.

The Cisco CSR 1000V can be deployed on Amazon Web Services (AWS) for public cloud solutions. Companies typically connect to their applications through a single VPN tunnel between their data center and AWS. Deploying the Cisco CSR in AWS opens up the potential for direct VPN access to AWS-hosted applications from campus and branch-office locations without back hauling through an existing data center. This design reduces latency, eliminates the need for expensive private WAN services, avoids the per-VPN-tunnel costs that Amazon charges, allows AWS hosted applications to participate in existing route-based VPN topologies, and lets enterprises own their network elements (which they use to enforce enterprise policies) within a public cloud infrastructure. This ownership helps the enterprise in different inter-cloud or cloudburst use cases described earlier in this chapter.

Amazon Web Service Solutions

These are the commonly used AWS solutions:

- **Amazon Elastic Compute Cloud (EC2)**—Amazon EC2 is a service that provides scalable computing resources in the Amazon cloud. The Amazon EC2 service offers flexible and scalable computing capacity that operators can spin up on demand. Amazon EC2 reduces the time for starting new server instances by providing a web portal that enables users to create, launch, and terminate server instances as needed and pay by the hour for active computing capacity usage.
- **Amazon Virtual Private Cloud (VPC)**—Amazon VPC is a cloud service that provides users a virtual private cloud by offering an isolated private section that exists

within Amazon’s public cloud. Amazon EC2 instances can run within a VPC, and enterprise customers can gain access to the EC2 instances over an IPsec VPN connection. Amazon VPC gives users the option of selecting which AWS resources are public facing and which are private, thus offering more granular control over security for instances running in AWS.

- **Amazon Simple Storage Service (S3)**—Amazon S3 is a service that provides users data storage infrastructure through web services interfaces (such as REST APIs). Through the web service interfaces, Amazon S3 offers simple storage and retrieval of data from anywhere on the web. Like Amazon EC2 service, with S3, users pay for the storage they actually use, which makes for a flexible and cost-effective object storage solution.

Routing in AWS Clouds

AWS offers a suite of networking capabilities that enable fundamental connectivity and traffic management for traffic going to and from applications hosted in the AWS cloud platform. EC2 offers two different platforms:

- **EC2-Classic**—This was introduced in the original release of Amazon EC2. All the Amazon Machine Instances (AMI) run in a single, flat network shared with other customers. All nodes running in EC2-Classic are on a shared network and are addressable to one another. There is no differentiation between public and private interfaces for AMI running in the EC2-Classic, as each machine instance has only one network interface. Each instance automatically receives a public IP address and can access the Internet directly.
- **EC2-VPC**—The AMIs launched in EC2-VPC run in a virtual private cloud (VPC) that is logically isolated from the user’s AWS account. Each instance in a VPC has a default elastic network interface associated with various attributes, such as multiple private IP addresses. One or more network interfaces can be attached to an instance during launch or added later from the EC2 console. Access to the Internet from AMI running in EC2-VPC is configurable and determined by the VPC policy.

Figure 6-20 shows a side-by-side comparison of EC2-Classic and EC2-VPC.

To understand the routing capabilities in AWS clouds, you must understand AWS VPC, the networking layers for Amazon EC2. These are the components of VPC:

- **Subnets**—A subnet is defined by AWS as a range of IP addresses in a VPC. When a VPC is created, the user can specify the setup of IP addresses for the VPC in the form of a Classless Inter-Domain Routing (CIDR) block, such as 172.16.0.0/16. A default subnet is a public subnet that is reachable from the Internet. Instances launched into a default subnet are each assigned a public IP address and a private IP address. Removing the route from the destination 0.0.0.0/0 can create a private subnet. Doing this prevents any EC2 instance running in that subnet from accessing the Internet.

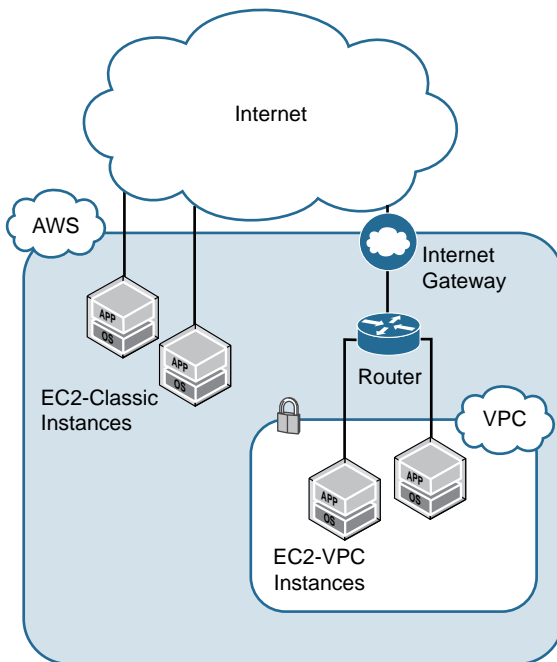


Figure 6-20 *Difference Between EC2-Classical and EC2-VPC*

- **Routing table**—Each subnet is associated with a single routing table that controls that subnet’s routing. The routing table contains a set of rules that define how traffic is forwarded within the VPC. Creating a VPC generates the main routing table automatically, enabling machine instances in the VPC to communicate with one another. A routing table can contain multiple subnets; however, only one routing table can be associated with each subnet.
- **Security group**—The AWS security group acts as a virtual firewall to protect virtual machine instances with inbound and outbound rules. Users can configure ports and protocols to open to traffic and from which source and to which destination. The security group operates at the instance level as the first layer of defense. It can be combined with network ACL, which operates at the subnet level, as a second layer of defense.
- **Elastic IP**—This is an Internet-routable public IP address associated with an AMI running in a VPC. It is assigned dynamically from a pool of EC2-VPC public IP addresses. The system allows fast failure convergence by allowing the elastic IP address to remap from one instance to another instance running in the same VPC upon the failure of the instance, thus minimizing impact to the end user experience. The elastic IP address, once it is assigned, is accessed through the Internet gateway of a VPC.
- **Internet gateway**—When instances are launched within a VPC, by default, they cannot communicate with the Internet. A user can enable Internet access by associ-

ating the VPC with an Internet gateway. The Internet gateway allows the instances to be reachable via the elastic IP address from the Internet.

- **Direct Connect**—Amazon Direct Connect is a network connection that provides a dedicated connectivity between enterprise and AWS cloud service. With AWS Direct Connect, an enterprise can establish a private link between the AWS cloud and the enterprise’s private data center. It provides consistent network performance over the Internet based connections. Dedicated Direct Connect addresses the privacy concerns related to data traversing the public Internet infrastructure.
- **Regions**—Amazon EC2 is hosted in multiple data center facilities in different geographic locations around the world. A region is an AWS data center facility location. Each Amazon EC2 region is completely independent of the others and is designed to be completely isolated from the other EC2 regions. EC2 instances have to be launched into a specific region. Locating EC2 instances close to end users reduces the access latency to the applications hosted there.
- **Availability zones**—Availability zones are isolated locations inside a region. The availability zones within a region are interconnected through low-latency network connections. By launching Amazon EC2 instances into separate availability zones, you can protect the applications from failure of a single availability zone. Figure 6-21 highlights the differences between EC2 CLASSIC and EC2 VPC.

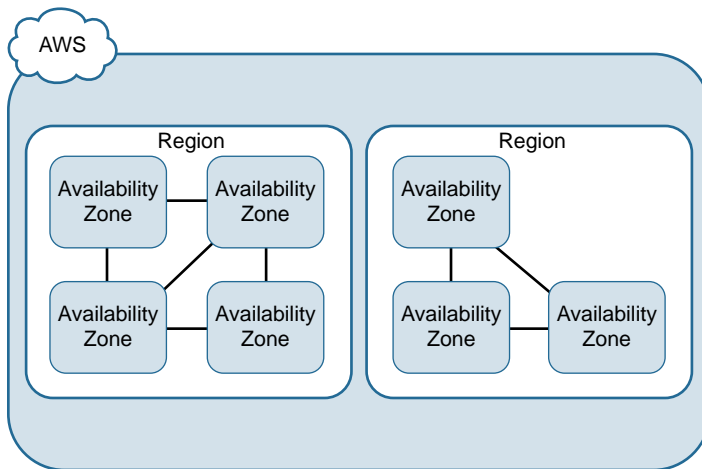


Figure 6-21 AWS Region and Availability Zone Concepts

Note While a VPC can span multiple availability zones within the AWS cloud, a subnet can only reside in a single zone and cannot span availability zones.

Amazon has designed its data center networks quite differently from the conventional enterprise approach. For one thing, a VLAN is not used as a Layer 2 segmentation

mechanism in Amazon cloud to avoid the 4096 VLAN scaling limitations associated with the technology. Instead, Amazon uses Layer 3 routing technology throughout its infrastructure, which means all traffic is forwarded based on IP addresses instead of Layer 2 MAC addresses. One way of looking at Amazon networking is that it is a completely flat network without hierarchy, which can be contradictory to traditional network design. The benefit of a flat network is that the configuration and setup process for virtual machine instances is greatly simplified. The entire process is automated to a greater extent than in the traditional hosting environment. In addition, because customers are not segmented into assigned VLANs, this easily allows for the growth and shrinking of computing instances based on seasonality or customer demand. Customers can easily request additional instances through the Amazon web portal and launch the new instances from the overall IP addresses in a region within minutes. The IP addresses for the new instances may be quite different from the others assigned previously, but because the network is flat and all traffic is forwarded based on IP addresses, the incongruent address assignment is not a problem.

Within AWS, cloud address assignment is not persistent to a customer. AWS dynamically assigns an IP address to an instance's virtual network interface (vNIC). The vNIC can have one or more private IP addresses and one public IP address, which can be automatically assigned. Having two IP addresses means that the instance can send and receive data traffic from within the AWS cloud network using the private IP address while accessible from the Internet with the public IP address.

Amazon VPC is confined to a single region and cannot span regions. However, within the region, a VPC can span multiple availability zones, and by launching instances into separate availability zones, you can protect applications from the failure of a single availability zone. Within an availability zone, you can create one or more subnets. Each subnet must reside entirely within one availability zone and cannot span zones. All subnets can route to one another by default.

When a VPC is created, it is automatically associated with a main route table. Initially there is only one entry in the main route table, which is the CIDR block address associated with the VPC. When new subnets are added in the VPC, they are implicitly associated with the main route table by default.

An Internet gateway can be added to a VPC to provide communication between the instances running in the VPC and the Internet. When an Internet gateway is attached to the VPC, it provides a target in the main routing table for routing to the Internet. In addition, the Internet gateway also performs the NAT function for instances that have been assigned public IP addresses or elastic IP addresses. These concepts are illustrated in Figure 6-22.

With the fundamental networking of AWS in hand, next we look into deploying the CSR in the AWS cloud network.

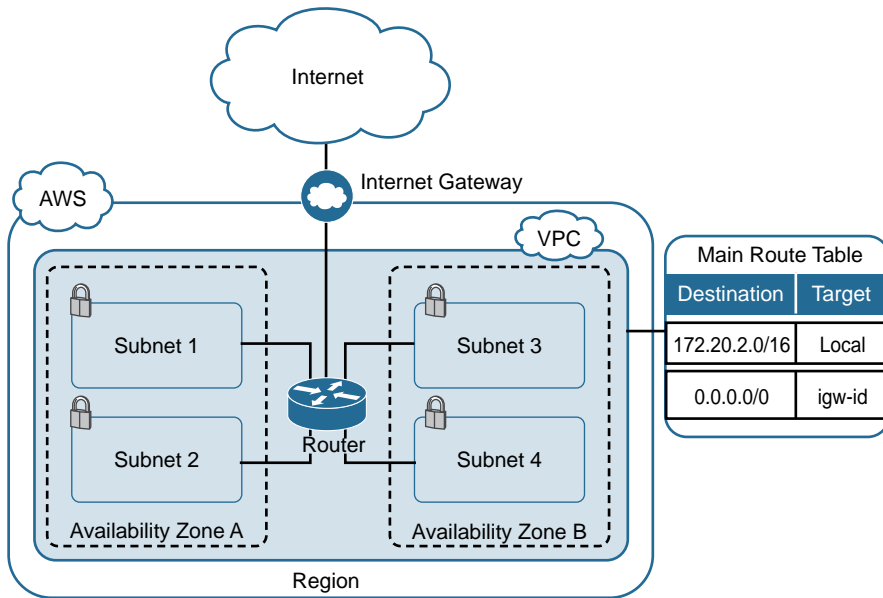


Figure 6-22 Amazon VPC Routing Topology

CSR 1000V Deployment in AWS

The Cisco CSR 1000V can be deployed on AWS for public and private cloud solutions. The CSR can be purchased and launched as an AMI from the AWS Marketplace. There are two license models currently available:

- **Hourly-Billed AMIs**—Users are billed for the hourly usage of the CSR 1000V instance. The hourly usage fee is in addition to the basic instance-type usage fee billed by AWS. AWS pays Cisco for CSR usage fees it collects. This model does not require any license to be purchased or installed.
- **Bring Your Own License AMI**—Users purchase software licenses directly from Cisco and launch the Bring Your Own License AMI from the AWS Marketplace. Users pay only for the basic instance-type fees. Once the CSR AMI is launched and deployed, users follow the standard Cisco software activation process to install the license.

From a design standpoint, CSR is logically placed between the VPC router and the machine instances running within the VPC, as shown in Figure 6-23.

However, the fact that the CSR is running as an AMI within a VPC and because of the networking property of AWS VPC, in reality the CSR sits in parallel to the machine instances, as illustrated in Figure 6-24. Therefore, a subnet route pointing to the CSR must be added to ensure traffic flow through CSR. Alternatively, the administrator must change the default gateway for each of the instances within the VPC to point to the CSR.

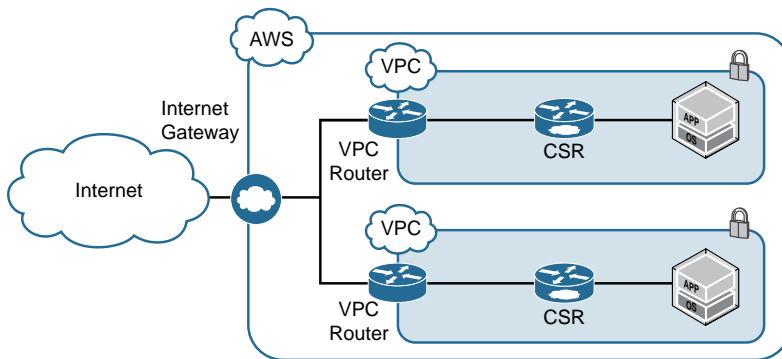


Figure 6-23 CSR Logical Placement Within AWS

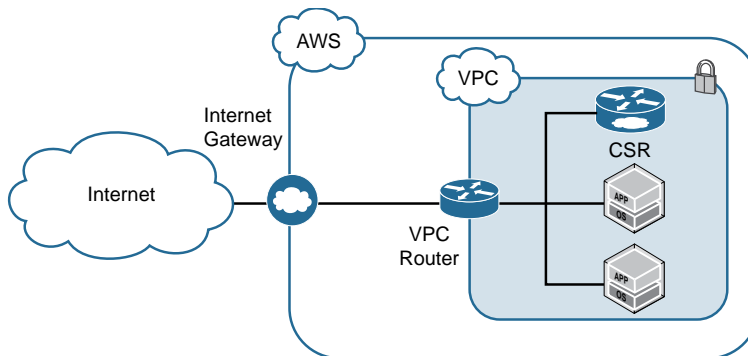


Figure 6-24 CSR Factual Placement in AWS VPC

Another property of an AWS VPC cloud network is that link-local multicast, and broadcast traffic are not supported. This restricts the use of First Hop Redundancy Protocol (FSRP), such as HSRP/VRRP, commonly leveraged by network engineers for high availability and the use of interior gateway protocols, such as OSPF and EIGRP, for route propagation. The workaround for some of these restrictions is the use of GRE encapsulation on the CSR. GRE encapsulation will transport the unsupported protocols. The GRE tunnel allows the CSR routers to exchange Bidirectional Forwarding Detection (BFD) protocol for rapid peer failure detection. When BFD detects a peer-down event, it triggers an Embedded Event Manager (EEM) script to modify the VPC route table, through the use of the AWS EC2 API, to redirect traffic around the failure. This use case can be expanded further to create tenants within a single VPC domain, as discussed earlier in this chapter, in the section “CSR 1000V as a Tenant Router.”

Instantiate a CSR in AWS

To launch the CSR, you must perform the following steps:

- Step 1.** Log in to AWS Marketplace at <https://aws.amazon.com/marketplace> and search for “Cisco csr,” as shown in Figure 6-25.

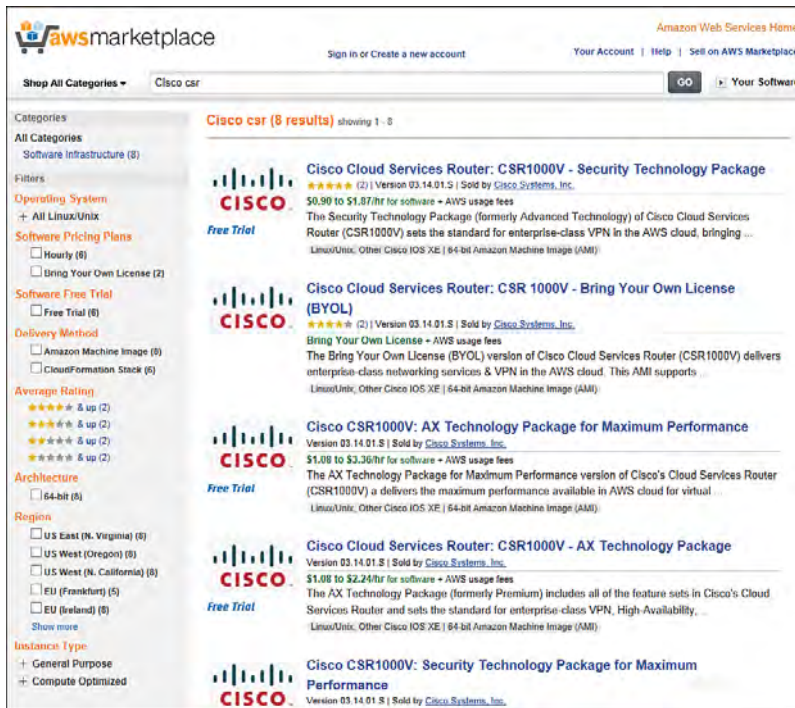


Figure 6-25 AWS Marketplace

Step 2. Select the Cisco CSR AMI for your deployment. You might have to log in or create a new account if you don't already have one. Choose one of the options for the AMI license:

- Hourly-Billed AMIs:
 - Cisco Cloud Services Router: CSR 1000V, Security Technology Package (formerly Advanced Technology)—Includes enterprise-rich security features, such as IPsec, DMVPN, FlexVPN, SSLVPN, and Zone-Based Firewall (ZBFW). The performance is based on the AMI instance type selected.
 - Cisco Cloud Services Router: CSR 1000V, AX Technology Package—Formerly the Premium package, the AX Technology Package includes all the features from the Security Technology Package plus features such as LISP and Application, Visibility, and Control (AVC), IPSLA, Performance Monitoring, and NBAR2. The performance is based on the AMI instance type selected.
 - “Maximum Performance” versions—These versions enable single root I/O virtualization (SR-IOV), which offers a direct I/O path for the AMI instance for higher and more consistent performance, as well as two times the performance with IMIX packets. Note that at

this writing, this is the only version that includes SR-IOV capability. In future releases of the CSR 1000V, the SR-IOV feature will be integrated into all versions.

- **CSR Direct Connect 1G/CSR Direction Connect Multi-Gig**—These AMIs are used for securing an enterprise-grade hybrid workload design with AWS Direct Connect circuits.
- **Cisco Cloud Services Router: CSR 1000V, Bring Your Own License (BYOL)**

Step 3. From the AWS EC2 launch page, select either the 1-Click Launch or the Manual Launch option to start the CSR AMI. Use the Manual Launch option for more granular control of the setup options. The following example uses the 1-Click Launch option to show the step-by-step procedure.

Note With 1-Click Launch, the user first creates the VPC, as well as security key pairs for remote SSH. Refer to the AWS documentation for how to set up VPC and key pairs. Also note that 1-Click Launch is currently not supported with BYOL AMIs.

Under VPC Settings, click the Set Up button (see Figure 6-26), which will take you to the VPC Settings page.

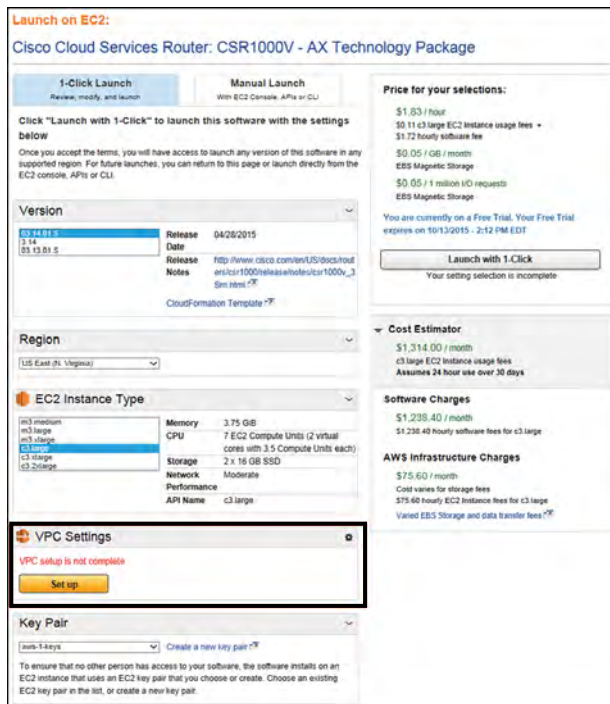


Figure 6-26 Cisco CSR 1-Click Launch Page

Step 4. On the VPC Settings screen, select the VPC for the CSR to use. Next, attach a public subnet and a private subnet to the CSR network interfaces. The security group for the public subnet is automatically created for the VPC. This security group is predefined, and users can change the security group settings after the AMI has launched. Click Done (see Figure 6-27) to return to the launch page.

Note By default, the security group is set up to allow SSH traffic. For IPsec deployment, IKE and ESP protocols have to be explicitly added to the security group.

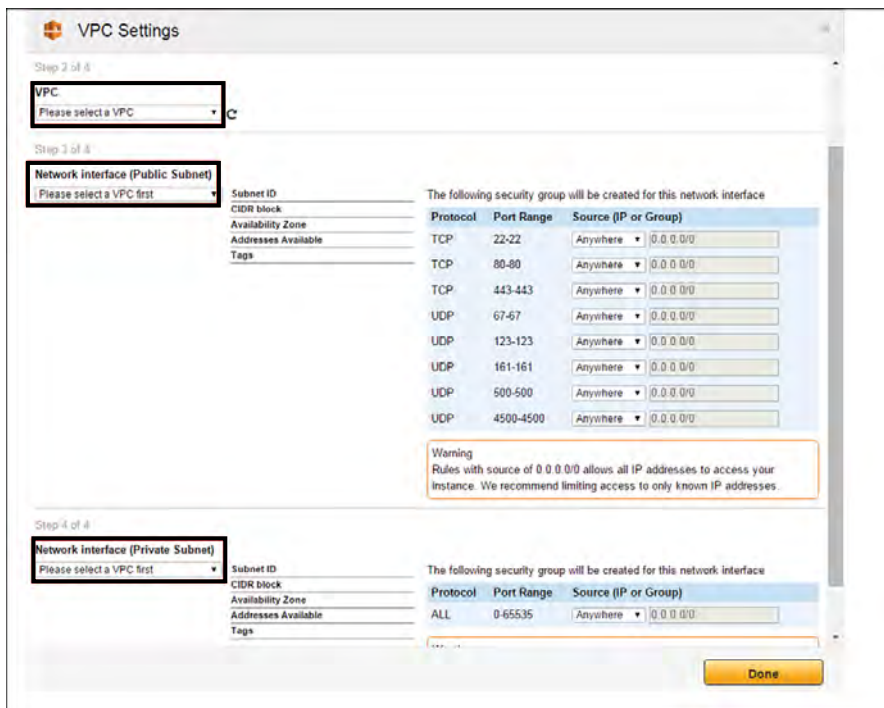


Figure 6-27 AWS VPC Settings Pop-up Window

Step 5. From the Launch page, go to the bottom and enter the Key Pair information. Choose from an existing key pair or select Create a New Key Pair if one does not exist and follow the directions. Next click the Launch with 1-Click button to start the CSR AMI instance (see Figure 6-28). It takes 5 to 10 minutes to deploy the instance and get the CSR fully up and running.

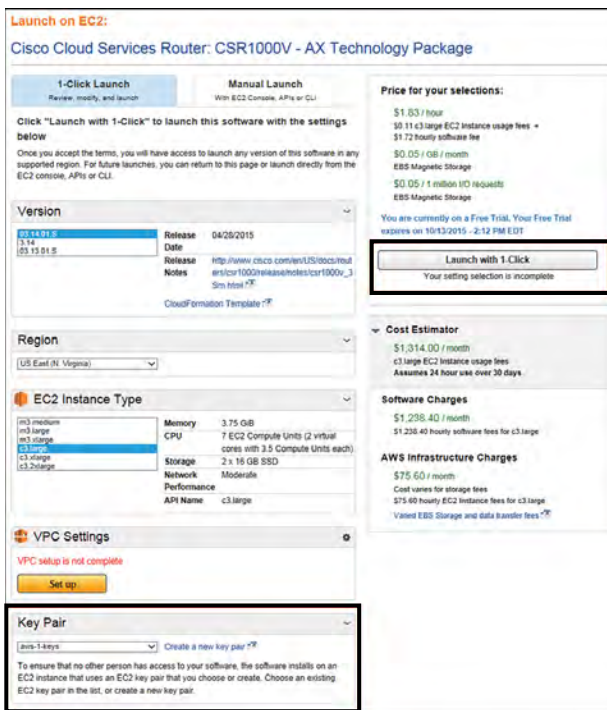


Figure 6-28 AWS Key Pair Selection and Launch with 1-Click

Step 6. To reach the CSR AMI through SSH for console access, enter the following procedures from a Unix shell command:

```
ssh -i <key-pair-pem-file-name> ec2-user@<public-ip-address or DNS-name>
```

Note For first-time login, use username `ec2-user` to access the instance.

These are the key considerations you need to review with AWS or any other public cloud provider:

- **Service solutions**—The type of service model that is offered by the cloud provider:
 - Scale-based solution (for example, Amazon EC2)
 - Multitenancy solution (for example, Amazon VPC)
 - Storage solutions (for example, Amazon S3)
- **Routing transport within public clouds**—Enterprise connectivity options to the cloud (for example, Internet Gateway, Direct Connect) and the need for extended tenancy options (for example, EC2-Classical and EC2-VPC).

- **Different cloud HA and models for geographical presence**—Provider-based HA (high availability) models based on enterprise HA requirements. This option requires proper study of the different models the cloud provider offers and how these models align with enterprise high-availability criteria for asset RPO (recovery point objective) and RTO (recovery time objective). Examples are regions and availability zones.

You must consider all these factors as you review any public cloud solution, whether from AWS, Google, Cisco, or Azure.

Summary

The CSR 1000V offers a lot of flexibility and capabilities such as overlay technologies, security gateway functionalities, and VRF-aware software infrastructure to provide a multitude of services inside of a multitenant data center. A data center architecture can leverage the CSR 1000V to effectively create security zone designs inside the data center and build preset logical communication between the zones within the data center owned by the enterprise.

With the increasing popularity of OpenStack as a cloud operating framework, there is an emphasis on the ability to leverage the different technologies found in a data center in a virtualized and programmable manner. The CSR 1000V offers enhancements to OpenStack networking capability beyond the built-in features and allows an enterprise to build a virtual data center at scale.

Now that you have read this chapter, you should have a clear understanding of using the CSR 1000V in a multitenant data center, with OpenStack, and in a public cloud environment. There are other use cases besides those covered in this chapter, and you can explore them as needed.

CSR in the SDN Framework

In Chapter 1, “Introduction to Cloud,” you got a brief introduction to software-defined networking (SDN). For a network administrator, SDN may mean automation and orchestration to simplify network operation. In this book you have also reviewed concepts of Network Functions Virtualization (NFV) and data center virtualization. You should understand how these two concepts overlap in an overall design for a cloud solution. The key thing to keep in mind is that in the cloud, you need automation and simplified orchestration. Reviewing the components of SDN in this chapter will help you understand the concept.

SDN creates abstract layers that cover three main components: controller, application, and data plane. This abstraction enables the creation of a customized network tailored to deliver the needs of the applications that run on it. It also provides an elastic data plane to users (an elastic data plane enables the addition of physical or virtual hardware resources on demand without the modification or disruption of the control plane). SDN can be categorized into three main functions as shown in Figure 7-1:

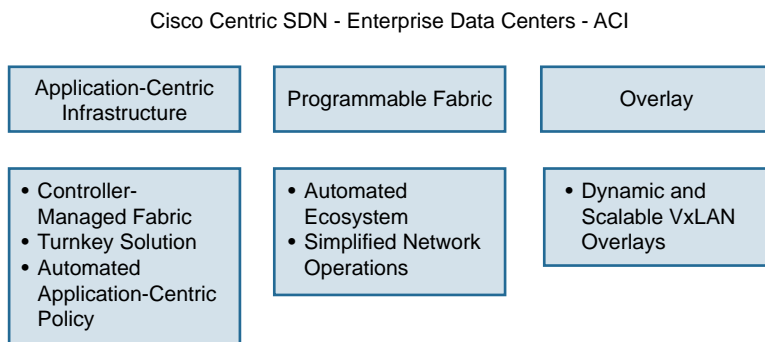


Figure 7-1 SDN Framework

- **Application-centric infrastructure**—Application programming interfaces (API) are exposed directly on network devices and service elements, based on OpenFlow elements.
- **Programmable fabric**—This fabric consists of controller and southbound protocol plugins (agents). A controller is a centralized unit used to configure and manage the agents. The agents reside in each node of the network providing data flow or services. Each agent has a set of routines and tools offering APIs that specify how the software components should interact with the controller. The communication between the controller and the agent enables the admin to control the whole network via a single controller. Having a single controller for the network infrastructure simplifies the admin’s adoption of automation and orchestration of services because there is one touchpoint for controlling the infrastructure platform.
- **Overlay**—A suite of overlay technologies deliver virtual network fabric between the nodes. The overlay network creates service-intelligent transport infrastructure over an elastic data plane. The overlay network deals with higher-level policy that is defined based on an application profile; the underlying infrastructure does the simple data forwarding between end nodes of the virtual infrastructure. Using an overlay also allows the admin to upgrade the physical infrastructure without impacting the design of the virtual overlay topology. The orchestration of the overlay is faster and simpler because it does not depend on configuring multiple hardware hops from source to destination. You have read about VxLAN, GRE, DMVPN, and MPLS VPNs, which are forms of overlay technologies that can be leveraged in an SDN environment.

SDN has multiple flavors, and its implementation varies based on vendor. In the Cisco product armada, multiple products are used to achieve abstraction of different layers, programmability, and orchestration via a central controller. For example, the Cisco Application Centric Infrastructure (ACI) is one such product solution. It is important to know what solutions are available and to understand their high-level components. Figure 7-2 shows out-of-the-box SDN for a data center that is vendor centric.

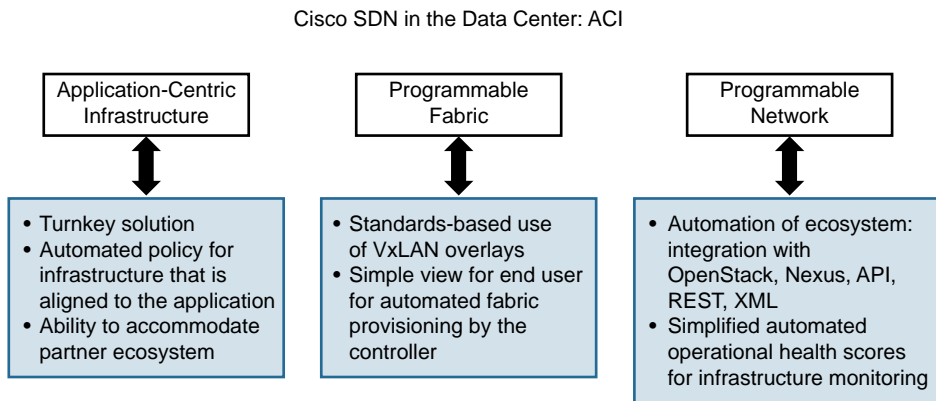


Figure 7-2 Cisco SDN in the Data Center

Another way to achieve this abstraction is with a combination of NFV elements and cloud management software, such as OpenStack. You have read in previous chapters about the history and various components of OpenStack.

OpenStack leverages NFV components in its framework to create an environment of vendor-agnostic cloud operations. OpenStack is focused on accelerating adoption of SDN by providing a robust SDN platform on which the industry can build and innovate. A centralized controller provides management of physical and virtual networks. The open source format makes it flexible for vendors like Cisco, IBM, and Red Hat to contribute to the project goals and platform architecture, as well as to its roadmap.

OpenStack brings together virtual networking approaches with Neutron. Please see Chapter 6, “CSR Cloud Deployment Scenarios,” for OpenStack and Neutron functionality, which provides virtual networking components to the cloud operating system.

This chapter explores the following:

- Step-by-step approach to deploying OpenStack
- Creating a tenant environment, using Neutron
- Adding the CSR 1000V as a tenant
- Replacing the functionality of Neutron with the CSR 1000V

Deploying OpenStack

This section covers deploying OpenStack in a proof-of-concept environment using Packstack installation. Packstack is a command-line tool that leverages Puppet modules for the rapid deployment of OpenStack on host machines. Packstack can be deployed interactively, using command-line prompts, or non-interactively, using defaults or a configuration file. This section illustrates deployment examples with Red Hat Enterprise Linux.

Packstack is suitable for deploying proof-of-concept installations where all controller services and virtual machines run on a single physical host. This is referred to as an all-in-one install. Packstack procedures are provided for an initial cloud deployment; the end result will depend on the method you choose and the parameters you define. Follow these steps to deploy OpenStack:

- Step 1.** Perform a minimum install of Red Hat Enterprise Linux on qualified server hardware.
- Step 2.** Change the interface names as needed for consistent and predictable network device naming for the network interfaces. These changes should make locating and differentiating the interfaces easier. In this case, you want to convert interface names like `enp9s0` to names like `eth0` by editing the `/etc/default/grub` file. Here is the first edit:

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$, ,g' /etc/system-release)"
```

```
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=auto
rd.lvm.lv=rhel/root rhgb quiet"
GRUB_DISABLE_RECOVERY="true"
```

Step 3. Change to the `GRUB_CMDLINE_LINUX` and add the following parameters:

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=auto
rd.lvm.lv=rhel/root rhgb quiet net.ifnames=0 biosdevname=0"
```

Step 4. Execute the following command to regenerate an updated `grub.cfg` file:

```
Grub2-mkconfig -o /boot/grub2/grub.cfg
```

Step 5. Change the names of the network scripts to `ethX`, where `X` is 0, 1, 2, and so on:

```
mv /etc/sysconfig/network-scripts/ifcfg-enp2s0 /etc/sysconfig/
network-scripts/ifcfg-eth0
```

Step 6. Update `NAME`, `BOOTPROTO=none`, `ONBOOT=yes` for each of the network scripts as follows:

■ `ifcfg-eth0` config:

```
TYPE=Ethernet
DEVICE=eth0
NAME=eth0
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
```

■ `ifcfg-eth1` config:

```
TYPE=Ethernet
DEVICE=eth1
NAME=eth1
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
```

Step 7. Reboot the system with the reconfiguration:

```
shutdown -r now
```

Note Interfaces on the “data network” of OpenStack are not mandated to have an IP address defined when tenant networks are VLAN based.

- Step 8.** Configure the IP addresses on interfaces that need to be used for OpenStack management and the external network (for example, `eth3` with IP addresses in the range `192.168.1.22/24`):

```
TYPE=Ethernet
DEVICE=eth3
NAME=eth3
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
IPADDR=192.168.1.22
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=8.8.8.8
DNS2=8.8.4.4
```

- Step 9.** Once the interfaces have been configured, shut down the server and restart the network service so the configuration can take effect. To do this, as root, issue the following command:

```
/etc/init.d/network restart
```

- Step 10.** OpenStack networking currently does not work on systems that have the NetworkManager service enabled. Use this configuration command, as root user, to disable the NetworkManager server:

```
systemctl stop NetworkManager
systemctl disable NetworkManager
```

- Step 11.** Start and enable the standard network service:

```
systemctl start network.service
systemctl enable network.service
```

- Step 12.** Each host machine deployed in OpenStack must register to Red Hat Subscription Management to receive updates from Red Hat Network. To allow Packstack to install OpenStack on each host, activate the correct subscriptions and repos as follows:

```
https://access.redhat.com/products/red-hat-enterprise-linux-openstack-platform/get-started
```

Subscribe the system via Red Hat Subscription Management and confirm that an OpenStack subscription is attached:

```
subscription-manager register
#Username : <userID>
#Password: <password>

subscription-manager subscribe --auto
subscription-manager list --consumed
```

If an OpenStack subscription is not attached immediately, see the documentation for manually attaching subscriptions.

Step 13. Clear the repositories that were initially set up and enable the ones needed for OpenStack:

```
subscription-manager repos --disable=*
subscription-manager repos --enable=rhel-7-server-rpms
subscription-manager repos --enable=rhel-7-server-rh-common-rpms
```

Step 14. Install the necessary yum packages, adjust the repository priority, and update:

```
yum repolist
yum install -y yum-plugin-priorities yum-utils
```

Figure 7-3 shows an example output from the yum install of OpenStack and the software collections associated with the operating system.

```
Loaded plugins: product-id, subscription-manager
rhel-7-server-opensstack-7.0-rpms           | 3.8 kB  00:00:00
rhel-7-server-rpms                         | 3.7 kB  00:00:00
rhel-server-rhsc1-7-rpms                   | 3.1 kB  00:00:00
(1/6): rhel-7-server-opensstack-7.0-rpms/7Server/x86_64/updateinfo | 48 kB  00:00:00
(2/6): rhel-7-server-opensstack-7.0-rpms/7Server/x86_64/group       | 130 B  00:00:00
(3/6): rhel-7-server-opensstack-6.0-rpms/7Server/x86_64/primary_db | 363 kB 00:00:00
(4/6): rhel-7-server-opensstack-7.0-rpms/7Server/x86_64/primary_db | 376 kB 00:00:00
(5/6): rhel-server-rhsc1-7-rpms/7Server/x86_64/primary_db         | 1.5 MB 00:00:01
(6/6): rhel-7-server-rpms/7Server/x86_64/primary_db               | 14 MB  00:00:16
(1/4): rhel-7-server-opensstack-6.0-rpms/7Server/x86_64/updateinfo | 101 kB 00:00:00
(2/4): rhel-server-rhsc1-7-rpms/7Server/x86_64/updateinfo         | 310 kB 00:00:00
(3/4): rhel-7-server-rpms/7Server/x86_64/group_gz                 | 133 kB 00:00:00
(4/4): rhel-7-server-rpms/7Server/x86_64/updateinfo               | 641 kB 00:00:00
No package yum-plugin-priorities available.
Resolving Dependencies
--> Running transaction check
--> Package yum-utils.noarch 0:1.1.31-29.e17 will be installed
--> Processing Dependency: python-kitchen for package: yum-utils-1.1.31-29.e17.noarch
--> Running transaction check
--> Package python-kitchen.noarch 0:1.1.1-5.e17 will be installed
--> Processing Dependency: python-chardet for package: python-kitchen-1.1.1-5.e17.noarch
--> Running transaction check
--> Package python-chardet.noarch 0:2.2.1-1.e17_1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package           Arch      Version      Repository      Size
=====
Installing:
yum-utils         noarch   1.1.31-29.e17  rhel-7-server-rpms  112 k
Installing for dependencies:
python-chardet   noarch   2.2.1-1.e17_1  rhel-7-server-rpms  227 k
python-kitchen   noarch   1.1.1-5.e17    rhel-7-server-rpms  266 k
=====
Transaction Summary
=====
Install 1 Package (+2 Dependent packages)
```

Figure 7-3 Sample Output from yum Install of OpenStack Packages

Execute the following commands to complete the yum package update:

```
#yum-config-manager --setopt="rhel-7-server-opensstack-7.0-rpms.
priority=1" --enable rhel-7-server-opensstack-7.0-rpms
#yum update -y
```


Figure 7-4 shows an example of output from enabling packages through the Red Hat Subscription Manager and making them available for the yum update utility.

```
[root@openstack-csr ~]# subscription-manager repos --enable rhel-7-server-openstack-7.0-rpms --enable
rhel-7-server-rpms
Repository 'rhel-7-server-rpms' is enabled for this system.
Repository 'rhel-7-server-openstack-7.0-rpms' is enabled for this system.
[root@openstack-csr ~]# yum-config-manager --setopt="rhel-7-server-openstack-7.0-rpms.priority=1" --
enable rhel-7-server-openstack-7.0-rpms

Loaded plugins: product-id, subscription-manager
rhel-7-server-openstack-7.0-rpms           | 3.8 kB  00:00:00
rhel-7-server-rpms                       | 3.7 kB  00:00:00
rhel-server-rhsc1-7-rpms                 | 3.1 kB  00:00:00
Resolving Dependencies
--> Running transaction check
=====
```

Figure 7-4 Sample Output from Enabling Packages from the Red Hat Repository to yum

Step 15. Enable optional extras and OpenStack repos:

```
subscription-manager repos --enable=rhel-7-server-optional-rpms
subscription-manager repos --enable=rhel-7-server-extras-rpms
```

Figure 7-5 shows an example of output from enabling the optional OpenStack repository.

```
Repository 'rhel-7-server-optional-rpms' is enabled for this system.
Repository 'rhel-7-server-extras-rpms' is enabled for this system.
```

Figure 7-5 Sample Output for Enabling the Optional OpenStack repository.

Step 16. Execute one of the following two options—but not both:

- Option A for RHEL OSP7:

```
subscription-manager repos --enable=rhel-7-server-openstack-7.0-rpms
```

- Option B for RDO (a community of people using and deploying OpenStack on CentOS, Fedora, and Red Hat Enterprise Linux-RHEL):

```
yum install http://rdoproject.org/repos/rdo-release.rpm
```

Step 17. Install other utilities:

```
yum install wget
#Is this ok [y/d/N]: y
yum install net-tools
#Is this ok [y/d/N]: y
yum install bind-utils
#Is this ok [y/d/N]: y
```

Step 18. Install Packstack:

```
yum install -y openstack-packstack
```

Figure 7-6 shows an output example of the Packstack installation through the yum utility.

| Dependencies Resolved | | | | |
|--|--------|-------------------------------------|----------------------------------|-------|
| Package | Arch | Version | Repository | Size |
| Installing: | | | | |
| openstack-packstack | noarch | 2015.1-0.11.dev1589.g1d6372f.el7ost | rhel-7-server-openstack-7.0-rpms | 231 k |
| Installing for dependencies: | | | | |
| PyYAML | x86_64 | 3.10-11.el7 | rhel-7-server-rpms | 153 k |
| jbigkit-libs | x86_64 | 2.0-11.el7 | rhel-7-server-rpms | 46 k |
| libjpeg-turbo | x86_64 | 1.2.90-5.el7 | rhel-7-server-rpms | 134 k |
| libtiff | x86_64 | 4.0.3-14.el7 | rhel-7-server-rpms | 167 k |
| libwebp | x86_64 | 0.3.0-3.el7 | rhel-7-server-rpms | 170 k |
| libyaml | x86_64 | 0.1.4-11.el7_0 | rhel-7-server-rpms | 55 k |
| openstack-packstack-puppet | noarch | 2015.1-0.11.dev1589.g1d6372f.el7ost | rhel-7-server-openstack-7.0-rpms | 20 k |
| openstack-puppet-modules | noarch | 2015.1.8-21.el7ost | rhel-7-server-openstack-7.0-rpms | 1.6 M |
| python-docutils | noarch | 0.12-0.2.20140510svn7747.el7ost | rhel-7-server-openstack-7.0-rpms | 1.5 M |
| python-netaddr | noarch | 0.7.12-1.el7ost | rhel-7-server-openstack-7.0-rpms | 1.3 M |
| python-pillow | x86_64 | 2.0.0-17.gitd1c6db8.el7 | rhel-7-server-rpms | 438 k |
| ruby | x86_64 | 2.0.0.598-25.el7_1 | rhel-7-server-rpms | 67 k |
| ruby-irb | noarch | 2.0.0.598-25.el7_1 | rhel-7-server-rpms | 88 k |
| ruby-libs | x86_64 | 2.0.0.598-25.el7_1 | rhel-7-server-rpms | 2.8 M |
| rubygem-bigdecimal | x86_64 | 1.2.0-25.el7_1 | rhel-7-server-rpms | 79 k |
| rubygem-io-console | x86_64 | 0.4.2-25.el7_1 | rhel-7-server-rpms | 50 k |
| rubygem-json | x86_64 | 1.7.7-25.el7_1 | rhel-7-server-rpms | 75 k |
| rubygem-psych | x86_64 | 2.0.0-25.el7_1 | rhel-7-server-rpms | 77 k |
| rubygem-rdoc | noarch | 4.0.0-25.el7_1 | rhel-7-server-rpms | 318 k |
| rubygems | noarch | 2.0.14-25.el7_1 | rhel-7-server-rpms | 212 k |
| Transaction Summary | | | | |
| Install 1 Package (+20 Dependent packages) | | | | |

Figure 7-6 Sample Output from the OpenStack Packstack Installation

Step 19. Generate a Packstack answer file:

```
packstack --gen-answer-file=answer_file.txt
```

Step 20. Edit the Packstack answer file as follows (see Appendix A, “Sample Answer File for Packstack”):

- Add the IP addresses of the computing nodes.
- Add physnet/bridge/interface information.
- Update the Neutron type driver info to be VLAN (instead of the default VxLAN).
- Update usernames and passwords (CONFIG_KEYSTONE_ADMIN_PW, CONFIG_KEYSTONE_DEMO_PW, CONFIG_DEFAULT_PASSWORD, and so on)
- Enable the HEAT/CIELOMETER/TEMPEST install.
- Enable the DEMO config.
- Set CONFIG_CINDER_VOLUMES_SIZE=1400G or as much disk space as can be spared from / on the control node.

Example 7-1 shows an abbreviated sample answer file for Packstack; see Appendix A for the entire output.

Example 7-1 *Sample Answer File for Packstack*

```
[general]
# Default password to be used everywhere (overridden by passwords set
# for individual services or users).
CONFIG_SSH_KEY=/root/.ssh/id_rsa.pub
CONFIG_DEFAULT_PASSWORD=Lab_P@sswd
CONFIG_MARIADB_INSTALL=y
# Install OpenStack Image Service (glance)
CONFIG_GLANCE_INSTALL=y
# Install OpenStack Block Storage (cinder)
CONFIG_CINDER_INSTALL=y
# Install OpenStack Shared File System (manila)
CONFIG_MANILA_INSTALL=y
# Install OpenStack Compute (nova)
CONFIG_NOVA_INSTALL=y
# Install OpenStack Networking (neutron)
CONFIG_NEUTRON_INSTALL=y
# Install OpenStack Dashboard (horizon)
CONFIG_HORIZON_INSTALL=y
# Install OpenStack Object Storage (swift)
CONFIG_SWIFT_INSTALL=y
# Install OpenStack Metering (ceilometer)
CONFIG_CEILOMETER_INSTALL=y
# Install OpenStack Orchestration (heat)
CONFIG_HEAT_INSTALL=y
CONFIG_SAHARA_INSTALL=n
CONFIG_TROVE_INSTALL=n
CONFIG_IRONIC_INSTALL=n
CONFIG_CLIENT_INSTALL=y
CONFIG_NAGIOS_INSTALL=y
CONFIG_DEBUG_MODE=n
CONFIG_AMQP_NSS_CERTDB_PW=Lab_P@sswd
CONFIG_AMQP_AUTH_USER=amqp_user
CONFIG_AMQP_AUTH_PASSWORD=Lab_P@sswd
CONFIG_KEYSTONE_ADMIN_USERNAME=admin
CONFIG_KEYSTONE_ADMIN_PW=Lab_P@sswd
CONFIG_KEYSTONE_DEMO_PW=Lab_P@sswd
# Size of Block Storage volumes group. the size of the volume group
# will restrict the amount of disk space that you can expose to
# Compute instances.
CONFIG_CINDER_VOLUMES_SIZE=1400G
CONFIG_MANILA_DB_PW=Lab_P@sswd
CONFIG_MANILA_KS_PW=Lab_P@sswd
CONFIG_MANILA_NETWORK_TYPE=neutron
CONFIG_IRONIC_DB_PW=Lab_P@sswd
CONFIG_IRONIC_KS_PW=Lab_P@sswd
# Private interface for flat DHCP on the Compute servers.
```

```

CONFIG_NOVA_COMPUTE_PRIVIF=eth1.10
# The name of the Open vSwitch bridge (or empty for linuxbridge) for
# the OpenStack Networking L3 agent to use for external traffic.
# Specify 'provider' if you intend to use a provider network to handle
# external traffic.
CONFIG_NEUTRON_L3_EXT_BRIDGE=br-ex
# Specify 'y' to install OpenStack Networking's Load-Balancing-
# as-a-Service (LBaaS)
CONFIG_LBAAS_INSTALL=y
# Specify 'y' to install OpenStack Networking's L3 Metering agent
CONFIG_NEUTRON_METERING_AGENT_INSTALL=y
# Specify 'y' to configure OpenStack Networking's Firewall-
# as-a-Service (FWaaS)
CONFIG_NEUTRON_FWAAS=y
# list of network-type driver entry points to be
# loaded from the neutron.ml2.type_drivers namespace
CONFIG_NEUTRON_ML2_TYPE_DRIVERS=local,vlan,flat,gre,vxlan
# Network types to allocate as tenant networks (local, vlan, gre, vxlan)
CONFIG_NEUTRON_ML2_TENANT_NETWORK_TYPES=vlan
CONFIG_NEUTRON_ML2_MECHANISM_DRIVERS=openvswitch
CONFIG_NEUTRON_ML2_FLAT_NETWORKS=*
# list of <physical_network>:<vlan_min>:<vlan_max> or
# <physical_network> specifying physical_network names usable for VLAN
# provider and tenant networks, as well as ranges of VLAN tags on each
# available for allocation to tenant networks.
CONFIG_NEUTRON_ML2_VLAN_RANGES=physnet1:2:4094
# list of <vni_min>:<vni_max> tuples enumerating ranges of
# VXLAN VNI IDs that are available for tenant network allocation.
CONFIG_NEUTRON_ML2_VNI_RANGES=5001:10000
CONFIG_NEUTRON_L2_AGENT=openvswitch
CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS=physnet1:br-data
# list of colon-separated Open vSwitch <bridge>:<interface> pairs.
# The interface will be added to the associated bridge. If you desire
# the bridge to be persistent a value must be added to this directive,
# also CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS must be set in order to
# create the proper port.CONFIG_NEUTRON_OVS_BRIDGE_IFACES=br-data:eth1.10
CONFIG_NEUTRON_OVS_VXLAN_UDP_PORT=4789
CONFIG_HEAT_DB_PW=Lab_P@sswd
CONFIG_HEAT_KS_PW=Lab_P@sswd
CONFIG_HEAT_DOMAIN_ADMIN=heat_admin
CONFIG_HEAT_DOMAIN_PASSWORD=Lab_P@sswd
# Specify 'y' to provision for demo usage and testing
CONFIG_PROVISION_DEMO=y
# Specify 'y' to configure the OpenStack Integration Test Suite
# (tempest) for testing.
CONFIG_PROVISION_TEMPEST=y
CONFIG_PROVISION_DEMO_FLOATRANGE=172.24.4.224/28

```

```
CONFIG_PROVISION_TEMPEST_USER=admin
CONFIG_PROVISION_TEMPEST_USER_PW=Lab_P@sswd
CONFIG_SAHARA_DB_PW=Lab_P@sswd
CONFIG_SAHARA_KS_PW=Lab_P@sswd
CONFIG_TROVE_DB_PW=Lab_P@sswd
CONFIG_TROVE_KS_PW=Lab_P@sswd
CONFIG_TROVE_NOVA_USER=admin
CONFIG_TROVE_NOVA_TENANT=services
CONFIG_TROVE_NOVA_PW=Lab_P@sswd
```

Step 21. Run Packstack with the answer file:

```
packstack --answer-file=answer_file.txt
```

Deployment time can be significant; Packstack provides continuous updates indicating which manifests are being deployed as it progresses. Once the process is completed, a confirmation message similar to that shown in Figure 7-7 is displayed.

```

Welcome to the Packstack setup utility

The installation log file is available at: /var/tmp/packstack/20151009-125530-
cJ0AeX/openstack-setup.log

Installing:
Clean Up [ DONE ]
Discovering ip protocol version [ DONE ]
Setting up ssh keys [ DONE ]
Preparing servers [ DONE ]
Preinstalling Puppet and discovering hosts' details [ DONE ]
Adding pre install manifest entries [ DONE ]
Setting up CACERT [ DONE ]
Adding AMQP manifest entries [ DONE ]
Adding MariaDB manifest entries [ DONE ]
Fixing Keystone LDAP config parameters to be undef if empty [ DONE ]
Adding Keystone manifest entries [ DONE ]
Adding Glance Keystone manifest entries [ DONE ]
Adding Glance manifest entries [ DONE ]
Adding Cinder Keystone manifest entries [ DONE ]
Checking if the Cinder server has a cinder-volumes wq [ DONE ]
Adding Cinder manifest entries [ DONE ]
Adding Nova API manifest entries [ DONE ]
Adding Nova Keystone manifest entries [ DONE ]
Adding Nova Cert manifest entries [ DONE ]
Adding Nova Conductor manifest entries [ DONE ]
Creating ssh keys for Nova migration [ DONE ]
Gathering ssh host keys for Nova migration [ DONE ]
Adding Nova Compute manifest entries [ DONE ]
Adding Nova Scheduler manifest entries [ DONE ]
Adding Nova VNC Proxy manifest entries [ DONE ]
Adding OpenStack Network-related Nova manifest entries [ DONE ]
Adding Nova Common manifest entries [ DONE ]
Adding Neutron FWaaS Agent manifest entries [ DONE ]
Adding Neutron LBaaS Agent manifest entries [ DONE ]
Adding Neutron API manifest entries [ DONE ]
Adding Neutron Keystone manifest entries [ DONE ]
Adding Neutron L3 manifest entries [ DONE ]
Adding Neutron L2 Agent manifest entries [ DONE ]
Adding Neutron DHCP Agent manifest entries [ DONE ]
Adding Neutron Metering Agent manifest entries [ DONE ]
Adding Neutron Metadata Agent manifest entries [ DONE ]

Checking if NetworkManager is enabled and running [ DONE ]
Adding OpenStack Client manifest entries [ DONE ]
Adding Horizon manifest entries [ DONE ]
Adding Swift Keystone manifest entries [ DONE ]
Adding Swift builder manifest entries [ DONE ]
Adding Swift proxy manifest entries [ DONE ]
Adding Swift storage manifest entries [ DONE ]
Adding Swift common manifest entries [ DONE ]
Adding Heat manifest entries [ DONE ]
Adding Provisioning Demo manifest entries [ DONE ]
Adding Provisioning Tempest manifest entries [ DONE ]
Adding Provisioning Glance manifest entries [ DONE ]
Adding MongoDB manifest entries [ DONE ]
Adding Redis manifest entries [ DONE ]
Adding Ceilometer manifest entries [ DONE ]
Adding Ceilometer Keystone manifest entries [ DONE ]
Adding Nagios server manifest entries [ DONE ]
Adding Nagios host manifest entries [ DONE ]
Adding post install manifest entries [ DONE ]
Copying Puppet modules and manifests
Applying 192.168.1.22_prescript.pp [ DONE ]
192.168.1.22_prescript.pp: [ DONE ]
Applying 192.168.1.22_amqp.pp [ DONE ]
192.168.1.22_amqp.pp: [ DONE ]
192.168.1.22_mariadb.pp: [ DONE ]
Applying 192.168.1.22_keystone.pp [ DONE ]
Applying 192.168.1.22_glance.pp [ DONE ]
Applying 192.168.1.22_cinder.pp [ DONE ]
192.168.1.22_keystone.pp: [ DONE ]
192.168.1.22_glance.pp: [ DONE ]
192.168.1.22_cinder.pp: [ DONE ]
Applying 192.168.1.22_api_nova.pp [ DONE ]
192.168.1.22_api_nova.pp: [ DONE ]
Applying 192.168.1.22_nova.pp [ DONE ]
Applying 192.168.1.22_neutron.pp [ DONE ]
Applying 192.168.1.22_osclient.pp [ DONE ]
Applying 192.168.1.22_horizon.pp [ DONE ]
192.168.1.22_nova.pp: [ DONE ]
192.168.1.22_neutron.pp: [ DONE ]
192.168.1.22_horizon.pp: [ DONE ]
192.168.1.22_osclient.pp: [ DONE ]

Applying 192.168.1.22_ring_swift.pp [ DONE ]
192.168.1.22_ring_swift.pp: [ DONE ]
Applying 192.168.1.22_swift.pp [ DONE ]
192.168.1.22_swift.pp: [ DONE ]
Applying 192.168.1.22_heat.pp [ DONE ]
192.168.1.22_heat.pp: [ DONE ]
Applying 192.168.1.22_provision_demo.pp [ DONE ]
Applying 192.168.1.22_provision_tempest.pp [ DONE ]
Applying 192.168.1.22_provision_glance [ DONE ]
192.168.1.22_provision_demo.pp: [ DONE ]
192.168.1.22_provision_glance: [ DONE ]
Applying 192.168.1.22_mongodb.pp [ DONE ]
Applying 192.168.1.22_redis.pp [ DONE ]
192.168.1.22_mongodb.pp: [ DONE ]
192.168.1.22_redis.pp: [ DONE ]
Applying 192.168.1.22_ceilometer.pp [ DONE ]
192.168.1.22_ceilometer.pp: [ DONE ]
Applying 192.168.1.22_nagios.pp [ DONE ]
Applying 192.168.1.22_nagios_nrpe.pp [ DONE ]
192.168.1.22_nagios.pp: [ DONE ]
192.168.1.22_nagios_nrpe.pp: [ DONE ]
Applying 192.168.1.22_postscript.pp [ DONE ]
192.168.1.22_postscript.pp: [ DONE ]
Applying Puppet manifests [ DONE ]
Finalizing [ DONE ]

**** Installation completed successfully ****

```

Figure 7-7 Sample Output from the OpenStack Installation Using Packstack

CSR as an OpenStack Tenant Deployment

This section provides the steps for instantiating CSR as an OpenStack tenant VM:

- Step 1.** Download the CSR 1000V qcow2 image from the Cisco.com website.
- Step 2.** Log in to the OpenStack dashboard (<http://<IP Address>/dashboard>) from a browser, as shown in Figure 7-8.

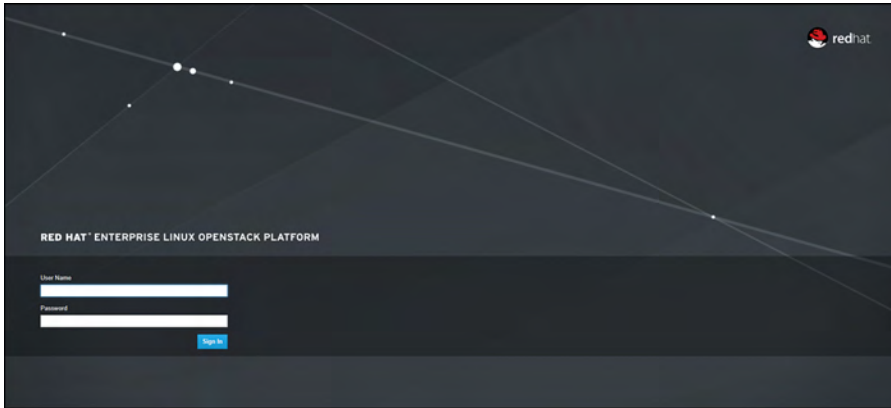


Figure 7-8 *OpenStack Dashboard Login*

- Step 3.** Upload the CSR 1000V image by navigating to Admin > Images > Create Image or using the `glance` CLI. Set both Minimum Disk and Minimum RAM to 0, as shown in Figure 7-9. Example 7-2 shows the use of the `glance` CLI to add a CSR 1000V image.

Figure 7-9 *OpenStack VM Image Creation*

Note To use the CLI, su to root on the OSP7 server and set source keystonec_admin before running the CLI.

Example 7-2 Example of Using the glance CLI to Add a CSR 1000V Image

```
source keystonec_admin

csr1kvImageSrc="/home/stack/csr_images/csr1000v-universalk9.03.16.00.S.155-3.S-ext.qcow2"
csr1kvImageName="csr1000v-3_16"
csr1kvDiskFormat="qcow2"
csr1kvContainerFormat="bare"
csr1kvGlanceExtraParams="--property hw_vif_model=virtio --property hw_disk_bus=virtio --property hw_cdrom_bus=ide"
tenantId='keystone tenant-list | grep " admin " | cut -f 2 -d'|''
glance image-create --name $csr1kvImageName --owner $tenantId --disk-format \
    $csr1kvDiskFormat --container-format $csr1kvContainerFormat \
    --file $csr1kvImageSrc $csr1kvGlanceExtraParams --is-public true
```

Use the glance CLI to verify that the CSR 1000V image is added correctly:

```
glance image-list
glance image-show csr1000v-3_16
```

- Step 4.** The CSR 1000V as an OpenStack tenant VM requires a minimum of two vCPUs and 4GB of RAM. Create a custom flavor for CSR with 4GB of RAM, zero disks, and two vCPUs by selecting Admin > Flavors > Create Flavor and making the settings shown in Figure 7-10.

The screenshot shows the 'Create Flavor' dialog in OpenStack. It has two tabs: 'Flavor Information' (selected) and 'Flavor Access'. The 'Flavor Information' tab contains several input fields: 'Name' (csr1000v-2vcpu-4gb), 'ID' (100), 'VCPUs' (2), 'RAM (MB)' (4096), 'Root Disk (GB)' (0), 'Ephemeral Disk (GB)' (0), and 'Swap Disk (MB)' (0). A note on the right states: 'Flavors define the sizes for RAM, disk, number of cores, and other resources and can be selected when users deploy instances.' At the bottom, there are 'Cancel' and 'Create Flavor' buttons.

Figure 7-10 OpenStack Custom VM Flavor

Alternatively, use the `nova` CLI to create a flavor with an ID of 100, 4GB of RAM, zero disks, and two vCPUs, as follows:

```
nova flavor-create csr.2vcpu.4gb 100 4096 0 2
nova flavor-list
nova flavor-show csr.2vcpu.4gb
```

- Step 5.** To create the public network and internal network within OpenStack, navigate to Project > Network > Network Topology > Create Network and fill out the form as shown in Figure 7-11. Then click Next.

The screenshot shows the 'Create Network' form in the OpenStack dashboard. The form is titled 'Create Network' and has a progress bar with three steps: 'Network' (active), 'Subnet *', and 'Subnet Detail'. The 'Network Name' field contains 'Public'. The 'Admin State' dropdown is set to 'UP'. A 'Next >' button is located at the bottom right of the form.

Figure 7-11 OpenStack Create Network Submenu

Enter the subnet name, network address, and gateway IP as shown in Figure 7-12, and then click Next.

The screenshot shows the 'Create Network' form in the OpenStack dashboard, specifically the 'Subnet' step. The 'Create Subnet' checkbox is checked. The 'Subnet Name' field contains 'Public-192.168.1.0/24'. The 'Network Address' field contains '192.168.1.0/24'. The 'IP Version' dropdown is set to 'IPv4'. The 'Gateway IP' field contains '192.168.1.1'. There is also a 'Disable Gateway' checkbox which is unchecked. A 'Next >' button is located at the bottom right of the form.

Figure 7-12 OpenStack Network Subnet Information

Enable DHCP as shown in Figure 7-13 and enter the DHCP pool address range in the Allocation Pools field. Fill in the DNS Name Servers and Host Routes fields if relevant.

Figure 7-13 OpenStack Network Subnet Detail

Alternatively, use the `neutron` CLI to create the subnets:

```
neutron net-create net-mgmt --provider:network-type localneutron
  subnet-create --name subnet-172-20-1 net-mgmt 172.20.1.0/24
```

```
neutron net-create net-internal --provider:network-type localneutron
  subnet-create --name subnet-172-16-1 net-int 172.16.1.0/24
```

To verify the subnets, use the following `neutron` command:

```
neutron net-listneutron net-show <net-name or uuid>neutron
  subnet-listneutron subnet-show <subnet-name or uuid>neutron
  port-listneutron port-show <port name or uuid>
```

- Step 6.** Create a `neutron` router and attach it to the internal and public network. From the OpenStack dashboard navigate to Project > Network > Network Topology > Create Router. The Create Router dialog appears, as shown in Figure 7-14.

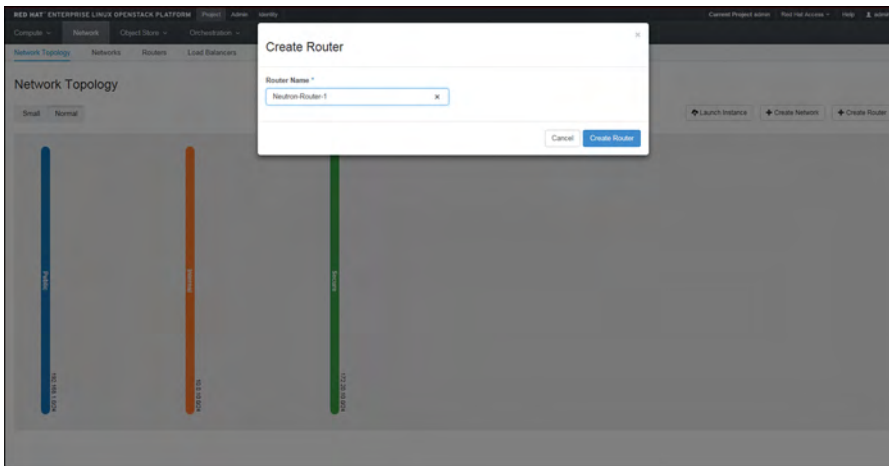


Figure 7-14 *OpenStack Create Router Dialog*

On the network topology page, click on the newly created `neutron` router icon and click `Add Interface`, as shown in Figure 7-15, to attach the router to the public and internal networks.

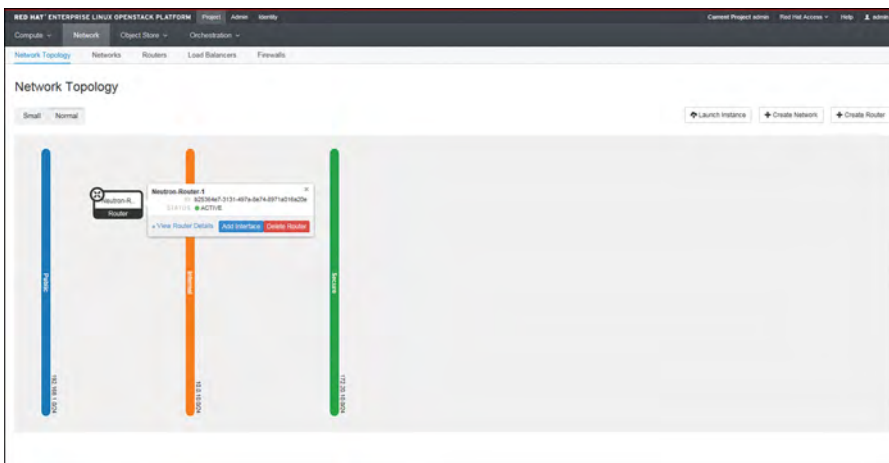


Figure 7-15 *OpenStack Attaching neutron Router Interfaces*

Here is an example of using the `neutron` CLI to create the router and attach it to the internal and public networks:

```
neutron router-create neutron-rtr-1
neutron router-interface-add neutron-rtr-1 subnet=subnet-10-20-1
neutron router-interface-add neutron-rtr-1 subnet=subnet-172-16-1
neutron router-gateway-set neutron-rtr-1 public
neutron router-port-list neutron-rtr-1
```

To verify that the neutron router is active and running, enter the following at the neutron CLI:

```
neutron route-list
neutron router-show <router name or uuid>
neutron port-list
neutron port-show <port name or uuid>
```

- Step 7.** Create a file for the CSR 1000V default configuration script when it boots up. Example 7-3 shows a sample of the configuration script.

Example 7-3 *Configuration Script Sample*

```
hostname csr1000v

line con 0

  logging synchronous
  transport preferred none

line vty 0 4
  login local
  transport preferred none
  transport input ssh

username stack priv 15 secret cisco
interface GigabitEthernet1
  ip address 10.11.12.2 255.255.255.0
  no shutdown
ip route 0.0.0.0 0.0.0.0 GigabitEthernet1 10.11.12.1
virtual-service csr_mgmt
  ip shared host-interface GigabitEthernet1
  activate

license accept end user agreement
license boot level premium
```

- Step 8.** Launch the CSR 1000V instance. From the dashboard, go to Project > Network > Network Topology > Launch Instance. Enter the instance name, choose the custom flavor created for the CSR 1000V, and select the CSR 1000V qcow2 image in Image Name (see Figure 7-16).

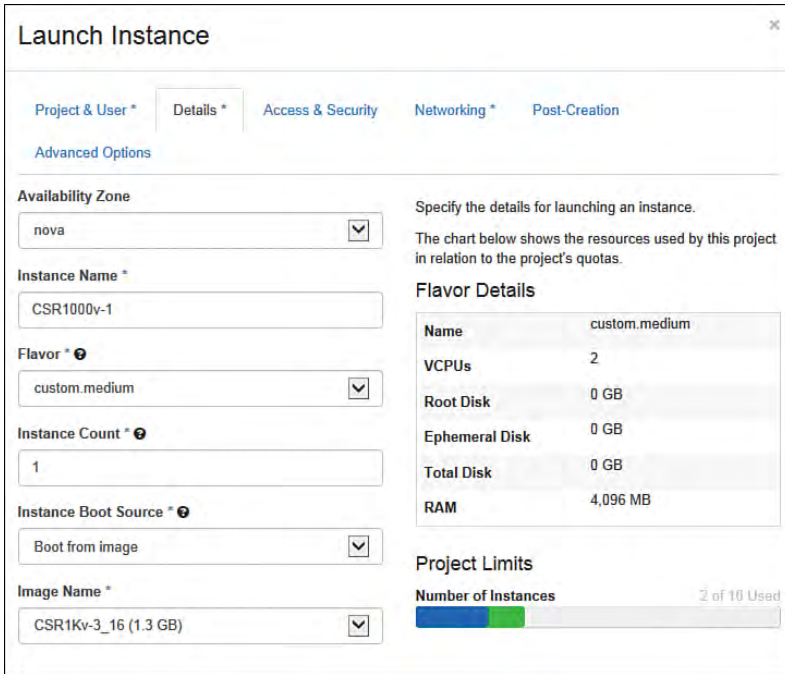


Figure 7-16 OpenStack Launching a VM Instance

On the Networking tab, select the networks the CSR 1000V will join (see Figure 7-17).

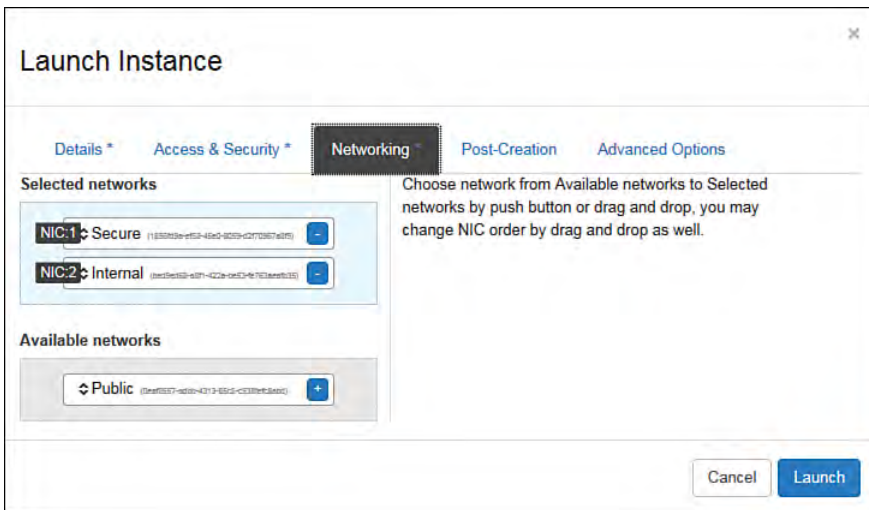


Figure 7-17 OpenStack VM Instance Network Detail

You can also launch the CSR 1000V tenant VM by using the following at the nova CLI:

```
nova boot --image csr1000v-3_16 --flavor csr.2vcpu.4gb
  --nic port-id=$MGMT_PORT_ID --nic port-id=$INTN_PORT_ID
  --nic port-id=$EXTN_PORT_ID --config-drive=true
  --file iosxe_config.txt=/opt/stack/iosxe_config.txt csr1000v-3_16
```

- Step 9.** Access the CSR 1000V console via the network topology page and click on the CSR 1000V instance to open the console access (see Figure 7-18). From there you can monitor the boot process of the VM.

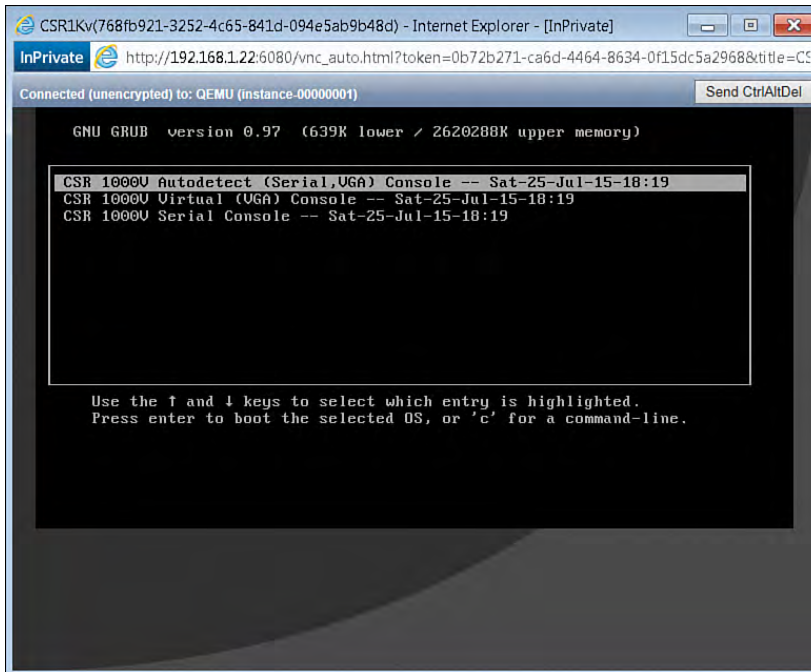


Figure 7-18 OpenStack CSR 1000V Splash Screen for grub

Instantiate CSR Plugin to OpenStack

OpenStack gives a cloud operator an open source platform for creating new services in a cloud environment. `nova` took care of the initial networking component of OpenStack, via `nova`'s own networking service, called `nova-network`. It has since then been superseded by the Neutron component introduced in the Folsom release. `nova-network` is still present today and can be used instead of Neutron. It is important to understand the difference between Neutron and `nova-network`.

`nova-network` has the following capabilities:

- Flat Network Manager offers Linux bridge for basic flat layer 2 network functions
- Flat DHCP Network Manager (`dnsmasq`) for IP address allocation and management
- Configuration of firewall policies and NAT in `IPTables`
- VLAN Network Manager supports VLAN Network mode for direct bridging, direct bridging with DHCP, and segments based on VLAN boundary

One of the limitations of `nova-network` is having three simple modes of networking that only support VLAN-based segmentation (with a max scale of 4094 VLANs). The `nova-network` stack lacks support for features like ACL and QoS. In addition, it is unable to leverage or integrate with third-party network vendors. Project Neutron incubated in April 2011 and was promoted to a core project at Folsom Summit in April 2012 to take care of these limitations. Neutron provides the ability to run multiple instances with an OpenStack-like rich feature set of APIs, plugin architecture for services, and a modular Layer 2. The plugin architecture of Neutron enables it to leverage more than one plugin at a given time.

Figure 7-19 shows the framework for the OpenStack Neutron plugin.

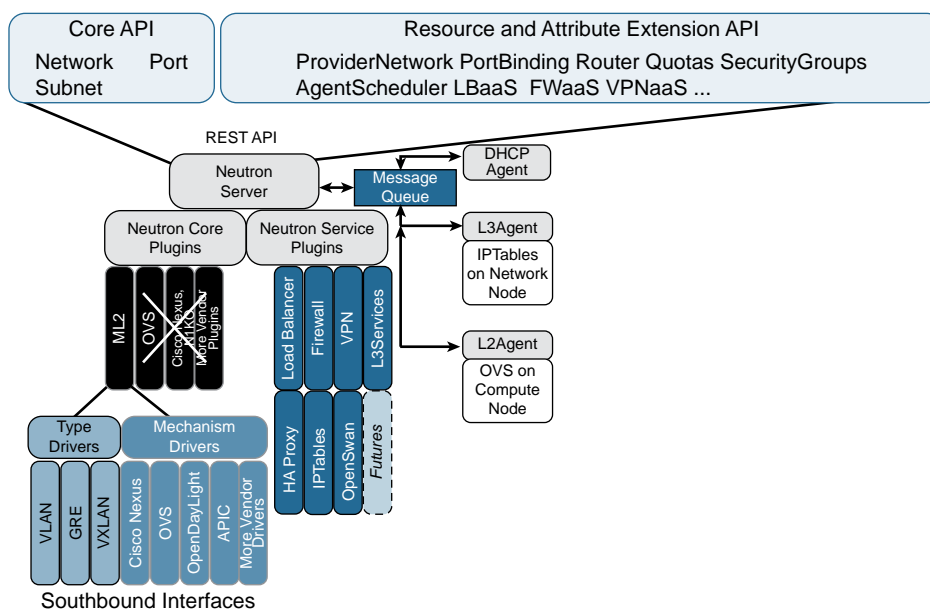


Figure 7-19 *Neutron Plugin Framework*

Figure 7-19 clearly shows a hierarchical structure for the plugins, as core and services plugins. You should be familiar with this diagram from Chapter 6, where we discussed OpenStack and CSR 1000V use cases. Core plugins offer Layer 2 and Layer 3

networking features such as VLAN, VxLAN, and GRE. The services plugin adds capabilities like firewalls, load balancers, and VPNs to the feature list that the OpenStack framework can support. There are limitations to the core and service plugins native to OpenStack. For example, the Neutron NAT function uses Linux IPTables for NAT, which has severe scale limitations. You can overcome these limitations by leveraging the CSR 1000V as an OpenStack plugin that offers robust Layer 3 forwarding and security service capabilities. OpenStack is a framework that can scale to different features and vendor-specific capabilities through the plugin architecture.

Chapter 6 covered how to deploy a CSR 1000V router as a neutron using plugins. Using devStack simplifies the process of replacing a neutron router with CSR 1000V. The plugins required for doing this can be retrieved from GitHub as detailed in the following steps (GitHub is a web-based Git hosting repository service):

Step 1. Install the Ubuntu server:

```
http://www.ubuntu.com/download/server/install-ubuntu-server
```

Step 2. Set these basic variables:

- If your setup is behind a proxy server, add the following environment variables in the `.bashrc` file:

```
export PROXY_HOST=proxyHost.cisco.com:80 export https_proxy=
https://$PROXY_HOST/
export http_proxy=http://$PROXY_HOST/
export ftp_proxy=http://$PROXY_HOST/
export HTTPS_PROXY=https://$PROXY_HOST/
export HTTP_PROXY=http://$PROXY_HOST/
export FTP_PROXY=http://$PROXY_HOST
export no_proxy="IpAddress"
```

- If your setup is behind a proxy server, switch `git://` to `https://` for `git` to work properly:

```
git config --global url."https://".insteadOf git://
git config --global https.insteadOf.git
```

- Disable `network-manager` if it is running via the following command:

```
sudo stop network-manager
```

Step 3. Download the CSR image `qcow2`.

Step 4. Download the OpenStack release `kilo devstack` from the OpenStack host server. After you install it, the directory `devstack` will be created under home directory:

```
cd ~/ git clone https://github.com/openstack-dev/devstack.git
```


Step 5. Set the devstack configuration file:

Create `localrc.conf` or `localrc` file under `~/devstack` directory and setup variables accordingly per your server setup.

Include the following in `localrc` or `local.conf` to download and install `networking-cisco`:

```
enable_plugin networking-cisco https://github.com/openstack/
networking-cisco.git master enable_service net-cisco
```

For CSR Routing-aaS, enter the following commands:

```
enable_service q-ciscorouter
enable_service ciscoconfigagent
```

For CSR VPN-aaS, enter the following. Ensure that CSR Routing-aaS is enabled.

```
enable_service cisco_vpn
```

For CSR FW-aaS, enter the following. Ensure that CSR Routing-aaS is enabled.

```
enable_service cisco-fwaas
```

The CSR 1000V now has the plugin in OpenStack to replace the neutron router that OpenStack uses by default. CSR 1000V (with a plugin running in OpenStack) offers a more feature-rich virtual routing functionality that compliments the OpenStack environment.

Summary

This chapter reviews the importance of SDN concepts in cloud environments. The SDN architecture has a basic framework that includes APIs, overlay, and controllers. The deployment and consumption of SDN depends on the user environments and also the hardware/software vendors involved. OpenStack provides cloud operators a framework and flexibility to manage a cloud environment. OpenStack also allows admins to leverage vendor-specific plugins to build new capability in the cloud infrastructure. You should now be comfortable deploying OpenStack and leveraging Neutron.

This page intentionally left blank

CSR 1000V Automation, Orchestration, and Troubleshooting

You already know to install a CSR on different hypervisor environments. In a large-scale deployment, manual processes increase the operational complexity of managing an NFV infrastructure. Automation in provisioning and management plays a major role in adopting NFV services in the data center or cloud environment. In this chapter you learn about the various tools available for simplifying orchestration and management of the CSR 1000V.

The terms *orchestration* and *automation* are sometimes used interchangeably. However, they are different. *Automation* involves achieving the completion of a single task, such as configuring a service on a router. *Orchestration*, on the other hand, involves automating a series of events to achieve a complete workflow or process.

Let us view this from a cloud perspective now. *Cloud automation* includes tasks that are required for deploying a cloud resource, such as spinning a routing instance (a virtual router, for example) in the cloud. Just this act of spinning a VM in the cloud should be considered automation. When you configure this instance, make it interact with other elements/instances and piece together a solution, it is *orchestration*.

Orchestrating a network solution can be broken down into the following high-level tasks:

- Creating an entity to service a networking requirement. In this chapter, this fundamental entity is a CSR. The CSR is the routing element that sits at the heart of the solution. The tool we discuss in this chapter for spawning a CSR is Cisco Build, Deploy, Execute OVF (BDEO).
- Managing the entity created using tools like Elastic Service Controller (ESC) that are popular for lifecycle management of VMs.
- Configuring the entity created. In this chapter, this configuration automation is done by Cisco Network Services Orchestrator (NSO).

With creation, life cycle management, and configuration automated, solutions can be designed to orchestrate a workflow and give turnkey solutions to customers. Virtual Managed Services (VMS) and Prime Network Services Controller (PNSC) automation solutions create a framework and use various tools to achieve service orchestration functionality.

Automation

The following sections review two common tools used for automation. The effort of automation varies based on the cloud use case that you intend to automate; the complexity of the tools also depends on this. You will learn about the BDEO tool, which is used to instantiate a single CSR 1000V instance, and the NSO tool, which is used to accomplish service chaining and network nodes.

BDEO

The BDEO tool is available to instantiate a CSR 1000V router. This tool is available when you download a CSR 1000V image from “CCO downloads.” As shown in Figure 8-1, the BDEO tool instantiates a CSR 1000V ISO image with the base configuration on the hypervisor. An administrator can leverage the BDEO tool to instantiate multiple CSR 1000V instances with the base configuration.

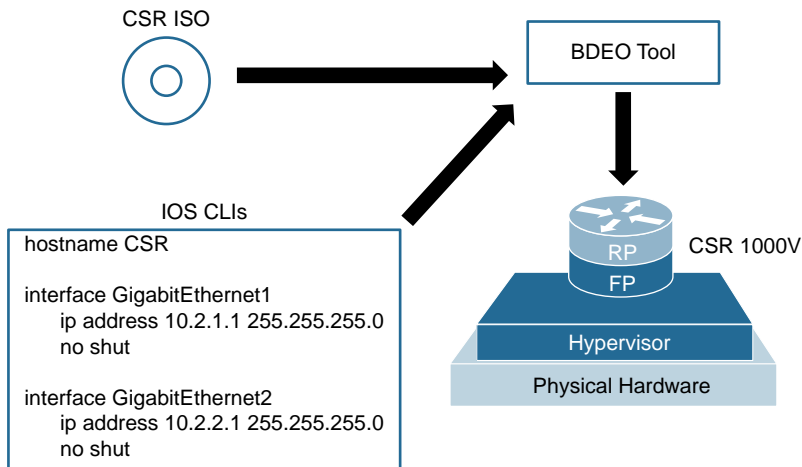


Figure 8-1 BDEO Overview

Note the following about the BDEO tool:

- It is currently supported only with VMware ESXi.
- It takes OVA (or ISO) as input. It outputs custom OVA, pre-provisioned with *basic IOS configuration* elements (management IP address, SSH, hostname, and so on). It is helpful for immediate provisioning.

- It is possible to apply a complete IOS config.txt, but you must deploy it via vCenter and cannot reference the host directly
- BDEO provides the intelligence to extract the config.info file and pass it to IOS, and it requires VMware's Open Virtualization Format (OVF) tool for deployment.

Note that for 3.9S1 and later versions of the CSR 1000V, the recommended tool for single/standalone installation is the Common OVF Tool (COT) instead of BDEO. You can use COD for editing the OVF of a virtual appliance with a focus on the Cisco CSR 1000V.

These are some of the key features of the BDEO tool:

- Can edit OVF hardware information (CPUs, RAM, NICs, configuration profiles, etc.)
- Can edit product description information in an OVF/OVA
- Can edit OVF environment properties
- Can embed a bootstrap configuration text file into an OVF/OVA
- Can deploy an OVF/OVA to an ESXi (VMware vSphere or vCenter) server to provision a new VM

NSO (Tail-f)

The NSO tool is the same as Tail-f, which has been mentioned in earlier chapters. Figure 8-2 shows the architecture of NSO.

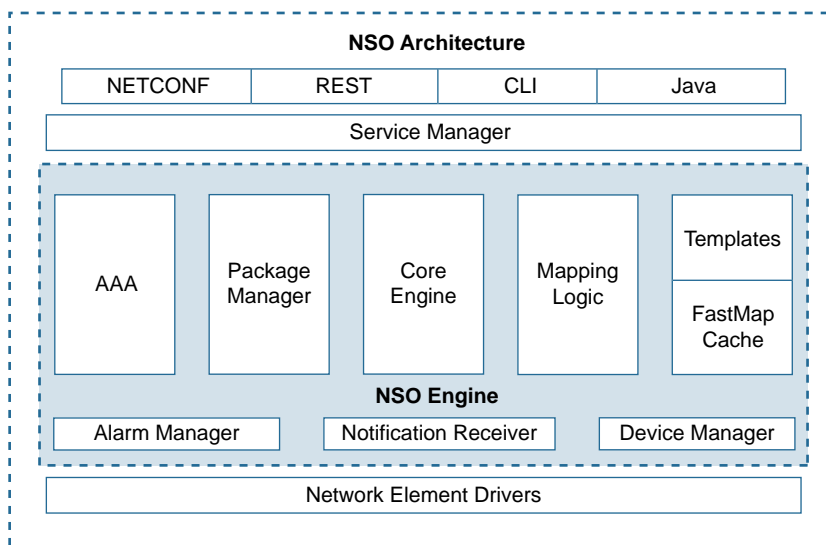


Figure 8-2 NSO Architecture

The following are some of the highlights of the NSO architecture, which is defined using a layered approach:

- The connectivity to the user interface (UI) or any interface stack for user-defined scripts for provisioning is handled with a rich set of tools, including NETCONF, REST, CLI, SNMP, and Java.
- The core engine described in Figure 8-2 is a key element of the NSO orchestration architecture. The core engine ties different blocks in the NSO architecture to execute a defined function.
- The NSO has a service manager that integrates the administrator-defined instructions to the orchestration engine.
- The AAA layer provides the engine role-based access to the devices and also to administrators accessing the device inventory.
- The package manager acts like a plugin that attaches itself to the core engine. The package manager provides flexibility for different use cases to be added to the orchestration profile, like GUI deployment for specific functionality that allows the pictorial representation of the orchestration and overview of health devices managed by NSO.
- The NSO engine is made up of the database with the mapping logic and fast map cache. Mapping logic and templates enable you to map intended service operations to network configuration required to implement the intended services. In other words, this is where a service intent is mapped to the elements that need to be modified to get the service implemented. Transactional integrity is maintained by a database that stores the configuration of each device on-boarded by the device manager. The unique thing that differentiates NSO from the rest of the orchestration tools is the capability to run non-real-time configuration event correlation of the new command function on the CDB (configuration database) baseline templates at fast Map cache. With FASTMAP you just need to specify the CREATE operation and not the REDEPLOY, UPDATE, and DELETE (RUD) operations. The RUD operation is automatically generated by the FASTMAP logic. This is done by having the NSO database view the actual as well as the intended device configurations for a particular service.
- The alarm manager manages alarms generated when there are errors pushing the configurations to end devices (via the NED) and from simulated results run in the Fast Map cache. The alarm manager can have a user-defined function written that takes action when a fault appears during a real-time deployment. The action can be a rollback, a log, or a priority-based log notification. The alarm manager is synced with the notification receiver that receives real-time notification.
- The device manager needs to be populated with a list of devices. This can be done manually or via automatic discovery.

- The network element drivers (NED) allow the tool to communicate with different device elements. This can also be vendor specific to collaborate with vendor capabilities—for example, if a YANG-based model is not supported, then a NED can be written to the vendor-specific CLI. These NEDs are part of the package manager that attaches with the core engine and maps access to devices. The NED also adds specific rules used in mapping engine services for the operation of NSO tool.
- The process of configuring upgrades in the NSO tool involves these steps:
 1. Provide provisioning instruction that is either UI driven (instruction communication through API, NETCONF, or through scripts) or user-provided CLI. The CLI of NSO abstracts the vendor-specific CLI to the admin. It translates the NSO CLI to the vendor-specific CLI in the NSO.
 2. Register the devices to be managed in the device manager.
 3. The NSO uses the device enlisted in the device manager to run the command in FastMap and mapping service for optimal configuration. Any false alarm is reviewed and assessed according to admin-defined rules.
 4. The appropriate NED pushes the optimal device configuration to the devices.
 5. The new configuration aligned to a new service is applied to a group of devices. The NSO communicates with all the devices that are required to participate in the service initiation provided by the admin. Any alarms on a single node are provided as feedback to the notification receiver and alarm manager. The action to the alarms is defined by the user-defined function, which can be a rollback to baseline configuration before the change or log the alarm and continue.
 6. After the new configuration is applied, the NSO adds the configuration to the CDB.

NSO is a very powerful tool and creates an abstraction for various functions. This tool can be modified and inserted in multiple solutions that can use different UIs, scripts for particular solutions, and NEDs as configuration management devices. For orchestration of NFV elements, ESC adds extra elasticity in provisioning and the life cycle that is helpful in cloud environments.

The deployment use case in NSO is explained in the simple three-tier layer shown in Figure 8-3.

The northbound APIs can invoke and provide input to a service model written in YANG. The service model provides a definition to the service functionality that needs to be deployed. This generic model can be used with multiple tenant instances to derive the same outcome. The device model is vendor-specific information that provides instructions for the device to function. This device model can cover CLI or YANG-based functions and interacts with the NED to communicate with the end devices.

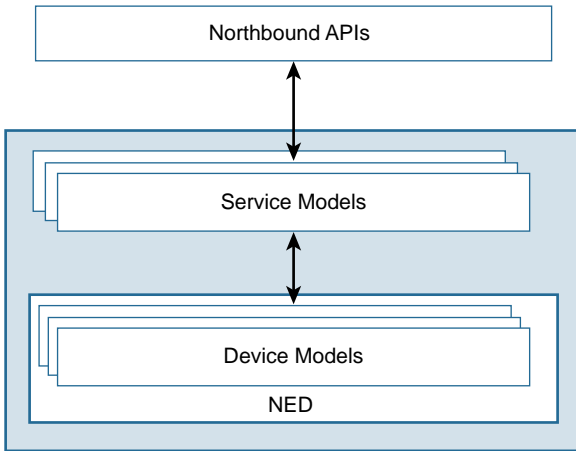


Figure 8-3 *NSO Deployment Framework*

NSO Example for NFV Orchestration with OpenStack (Service Chain)

Let's review, step by step, the NSO deployment framework illustrated in Figure 8-4. This workflow will help you visualize the deployment framework with an actual working model of NFV deployment in an OpenStack environment:

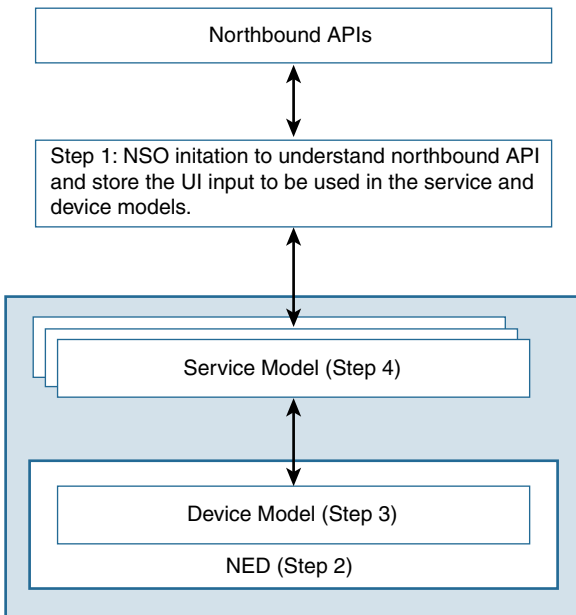


Figure 8-4 *NSO Deployment Use Case Flow*

- Step 1.** Define the service chain in NSO to get called by the northbound UI. This means designing the service, describing the intent (functionality that needs to be performed), and describing the elements needed to facilitate the request and input parameters that will be applied to the devices (speed, users, bandwidth, and so on).

Example 8-1 shows a simple base script written in YANG for service chaining.

Example 8-1 *Sample NSO Initiation to Understand Input to Be Used in the Service and Device Model Framework*

```
root@nso0:/nfv/poc/local/packages/serviceChain/src/yang# cat serviceChain.yang
module serviceChain {
  namespace "http://com/example/serviceChain";
  prefix serviceChain;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-common {
    prefix tailf;
  }
  import tailf-ncs {
    prefix ncs;
  }
  import resource-allocator {
    prefix ralloc;
  }

  typedef deviceref {
    type leafref {
      path "/ncs:devices/ncs:device/ncs:name";
    }
  }

  list dctopology {
    key dc-name;
    leaf dc-name {
      type string;
    }
    leaf dc-type {
      type enumeration {
        enum core;
        enum edge;
      }
    }
  }
}
```

```

leaf esc-os {
    type deviceref;
}
leaf openstack {
    type deviceref;
}
leaf vts-vc {
    type deviceref;
}

container zenoss {
    leaf ip {
        type inet:ipv4-address;
    }
    leaf community {
        type string;
    }
}

container vcenter {
    description "Configuration related to VMware, optional";
    presence vcenter;
    leaf vcenter {
        description "vCenter device for this DC";
        mandatory true;
        type deviceref;
    }
}

container esxi {
    container asa {
        leaf-list ip {
            description "List of ESXi hosts for provisioning ASAv";
            min-elements 1;
            type inet:ipv4-address;
        }
    }
    container csr {
        leaf-list ip {
            description "List of ESXi hosts for provisioning CSR";
            min-elements 1;
            type inet:ipv4-address;
        }
    }
}

leaf vswitch {

```

```
description "vSwitch where additional ports will be created";
mandatory true;
type string;
}

container mgmt-network {
description "Management interface configuration";
leaf mgmt-ip-pool {
description "resource-allocator ip-address-pool for management IPs";
type leafref {
path "/ralloc:resource-pools/ralloc:ip-address-pool/ralloc:name";
}
}

container mgmt-route-network {
description "Network to use when setting up route to mgmt gateway";
leaf network {
type inet:ipv4-address;
}
leaf netmask {
type inet:ipv4-address;
}
}

leaf datastore {
description "Datastore for deployed VNFs";
default datastore1;
type string;
}

leaf border-leaf {
description "Border leaf to be used as a gateway";
type inet:ipv4-address;
default 192.168.101.1;
}

augment /ncs:services {

list serviceChain {
description "Instantiate the service chaining function";

key name;
```

```
leaf name {
    tailf:info "Unique service id";
    tailf:cli-allow-range;
    type string;
}

uses ncs:service-data;
ncs:servicepoint serviceChain-servicepoint;

leaf tenant { // pass to ESC
    type string;
    mandatory true;
}

container status {
    config false;
    tailf:cdb-oper { tailf:persistent true; }
    leaf serviceChain-phase {
        type enumeration {
            enum tenant;
            enum network;
            enum day0;
            enum day1;
            enum finished;
        }
    }
}

list vm {
    key name;
    leaf name {
        type string;
    }
    leaf dc {
        type leafref {
            path "/serviceChain:dctopology/serviceChain:dc-name";
        }
        mandatory true;
    }
    leaf vim-type {
        type enumeration {
            enum openstack;
            enum vcenter;
        }
        mandatory true;
    }
}
```

```
}
leaf vm-type {
  type enumeration {
    enum asa;
    enum csr;
  }
}
container ip-pool {
  when "../vm-type = 'asa'";
  list pool {
    max-elements 1;
    min-elements 1;
    key "start end";
    leaf start {
      type inet:ipv4-address;
    }
    leaf end {
      type inet:ipv4-address;
    }
    leaf mask {
      mandatory true;
      type inet:ipv4-address;
    }
  }
}
container acl {
  when "../vm-type = 'csr'";
  list access-list {
    ordered-by user;
    key name;

    leaf name {
      type string;
    }

    leaf protocol {
      default ip;
      type enumeration {
        enum ip;
        enum icmp;
        enum tcp;
        enum udp;
      }
    }
  }
  leaf src-ip {
    mandatory true;
  }
}
```

```

        type inet:ipv4-address;
    }
    leaf src-mask {
        mandatory true;
        type inet:ipv4-address;
    }
    leaf src-port {
        when "../protocol = 'tcp' or ../protocol = 'udp'";
        type union {
            type uint16;
            type enumeration {
                enum any;
            }
        }
    }
    leaf dest-ip {
        mandatory true;
        type inet:ipv4-address;
    }
    leaf dest-mask {
        mandatory true;
        type inet:ipv4-address;
    }
    leaf dest-port {
        when "../protocol = 'tcp' or ../protocol = 'udp'";
        type union {
            type uint16;
            type enumeration {
                enum any;
            }
        }
    }
    leaf action {
        mandatory true;
        type enumeration {
            enum permit;
            enum deny;
        }
    }
}
list interface {
    max-elements 3;
    min-elements 3;
    key name;
    leaf name {

```

```
        type string;
    }
    leaf network-type {
        type enumeration {
            enum mgmt;
            enum private;
            enum external;
        }
    }
    leaf ip {
        type inet:ipv4-address;
    }
    leaf mask {
        type inet:ipv4-address;
    }
    leaf is-managed {
        type empty;
    }
    leaf network-name {
        type string;
    }
    leaf subnet-prefix {
        when "boolean(..is-managed)";
        type inet:ip-prefix;
    }
    leaf gateway-ip {
        when "boolean(..is-managed)";
        type inet:ip-address;
    }
} //interface
} // vm

tailf:action remove-service {
    tailf:info "Remove service";
    tailf:actionpoint serviceChain-remove-service;
    output {
        leaf success {
            type boolean;
        }
        leaf message {
            type string;
            description "Free format message.";
        }
    }
}
```

```

    } //serviceChain
  }

  augment "/ncs:devices/ncs:device" {
    leaf transition-workaround-ready {
      description "Workaround for checking readiness of device - transition package
        is weird after redeploy";
      config false;

      tailf:cdb-oper {
        tailf:persistent true;
      }
      type boolean;
    }
  }
}

```

This model provides basic elements needed for service chaining and defines the parameters of the VNF component and base configuration that can be orchestrated from the user using REST API or XML structure. Using this script, the NSO captures the functionality of the service chaining elements. Translating this to vendor-specific configuration needs special templates that tie to the NED (vendor-specific attributes). Multiple models can use these templates. The templates are vendor/device specific and are not covered in this book. This base template provides NSO a framework that enables it to receive inputs from the user interface.

- Step 2.** NSO should have the appropriate NEDs to communicate with the infrastructure southbound. The command `ls -l $NCS_APP/packages` helps verify the installation of the packages for the NEDs, as shown in Example 8-2.

Example 8-2 *Sample of Installed NED Verification*

```

ls -l $NCS_APP/packages/
cisco-asa
cisco-ios
cisco-iosxr
cisco-vpp
  CSR specific packages
##find cisco IOS
cisco-ios
cisco-ios/src
cisco-ios/src/ncsc-out
cisco-ios/src/ncsc-out/modules
cisco-ios/src/ncsc-out/modules/fixs
cisco-ios/src/ncsc-out/modules/fixs/tailf-ned-cisco-ios.fixs
cisco-ios/src/ncsc-out/modules/fixs/tailf-ned-cisco-ios-stats.fixs

```



```

cisco-ios/src/ncsc-out/modules/fixs/taif-ned-cisco-ios-id.fxs
cisco-ios/src/ncsc-out/modules/yang
cisco-ios/src/ncsc-out/modules/yang/taif-ned-cisco-ios-id.yang
cisco-ios/src/ncsc-out/modules/yang/taif-ned-cisco-ios-stats.yang
cisco-ios/src/ncsc-out/modules/yang/taif-ned-cisco-ios.yang
...

```

Step 3. Onboard VNF descriptors detail the characteristics of the VNF, such as capabilities (like firewall or router), resources required (CPU, storage, and so on), and artifacts to instantiate it (disk image). This is the device package in the deployment hierarchy. Example 8-3 provides an example of VNF descriptor definition. In this example, the ASA and CSR are both defined as VNF descriptors. These two instances will be reviewed in later service chaining steps.

Example 8-3 *Sample Definition of VNF Descriptors*

```

mano {
+   vnfd ASA941 {
+     version 9.4.1;
+     connection-points inside {
+       flavour small;
+       vdu ASA;
+       nic-id 2;
+     }
+     connection-points outside {
+       flavour small;
+       vdu ASA;
+       nic-id 1;
+     }
+     flavours small {
+       vdus ASA {
+         vm-spec {
+           pkg-uri http://192.168.100.16/nfv/qcow/asa941-200.qcow2;
+           disk-format qcow2;
+         }
+         storage root-disk {
+           size-gb 10;
+           storage-type root;
+         }
+         cpu {
+           num-vpu 2;
+         }
+         memory {
+           total-memory-gb 4;
+         }
+       }
+     }
+   }

```

```

+         device-type {
+             cli {
+                 ned-id cisco-asa;
+             }
+         }
+         authgroup asa;
+         day0 {
+             source-url http://192.168.100.16/nfv/day0/ASA_941_day0.txt;
+             destination-file day0-config;
+         }
+     }
+ }
+ vnfms {
+     esc esc0;
+ }
+ }
+ vnf CSR313 {
+     version 3.13.01;
+     connection-points left {
+         flavour small;
+         vdu CSR;
+         nic-id 1;
+     }
+     connection-points right {
+         flavour small;
+         vdu CSR;
+         nic-id 2;
+     }
+     flavours small {
+         vdus CSR {
+             vm-spec {
+                 pkg-uri http://192.168.100.16/nfv/qcow/csr1000v-universalk
+                     9.03.13.01.S.154-3.S1-ext.qcow2;
+                 disk-format qcow2;
+             }
+             storage root-disk {
+                 size-gb 8;
+                 storage-type root;
+             }
+             cpu {
+                 num-vpu 1;
+             }
+             memory {
+                 total-memory-gb 3;
+             }
+             device-type {

```

```
+         cli {
+             ned-id cisco-ios;
+         }
+     }
+     authgroup csr;
+     day0 {
+         source-url http://192.168.100.16/nfv/day0/CSR_day0.txt;
+         destination-file iosxe_config.txt;
+     }
+     disk-bus virtio;
+     serial-console true;
+     e1000-net true;
+ }
+ }
+ vnf {
+     esc esc0;
+ }
+ }
+ management-ip-pool net_osmgt;
+ nsd advanced {
+     connection-points a {
+         member-vnf fw;
+         vnf-connection-point inside;
+     }
+     connection-points z {
+         member-vnf router;
+         vnf-connection-point right;
+     }
+     flavours small {
+         member-vnfs fw {
+             vnf ASA941;
+             flavour small;
+             vdu ASA;
+         }
+         member-vnfs router {
+             vnf CSR313;
+             flavour small;
+             vdu CSR;
+         }
+     }
+ }
+ }
+ nsd basic {
+     connection-points a {
+         member-vnf router;
+         vnf-connection-point left;
+     }
+ }
```



```

+         p-connection-point a;
+         ce-connection-point z;
+         vnf esc0;                                << VNFM that will instantiate
the VNF
+     }
+     ce-interface GigabitEthernet0/1;            << From here on, specify L3VPN
configurations
+         ip-network 172.14.1.0/24;
+         bandwidth 50000;
+     }
+     endpoint main-office {                       << Second endpoint, the main
office - Provider
+         ce-device ce4;
+         ce-interface GigabitEthernet0/1;
+         ip-network 172.15.1.0/24;
+         bandwidth 60000;
+     }
+ }
}

```

Once you commit step 4 on the NSO, the model is executed from the NSO. The NSO uses the ESC to spawn the VNF elements.

The following steps illustrate script execution phases:

- Before executing Step 4, no service is associated with the tenants (see Figure 8-5).



Figure 8-5 *OpenStack Tenant*

- Figure 8-6 shows the OpenStack tenant after execution of the script.

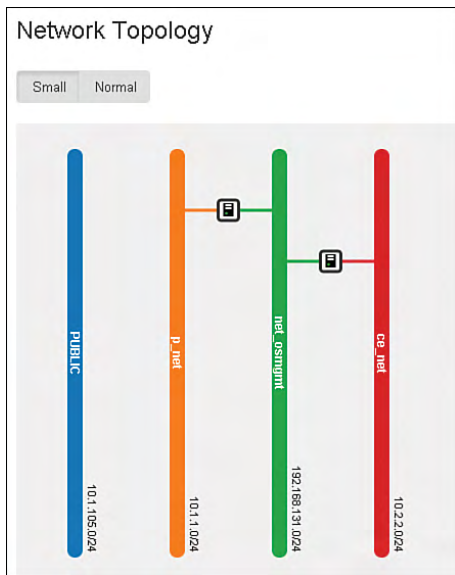


Figure 8-6 OpenStack Tenant After Execution

The following service chaining components are added after execution of the script:

- ASA (see Figure 8-7)

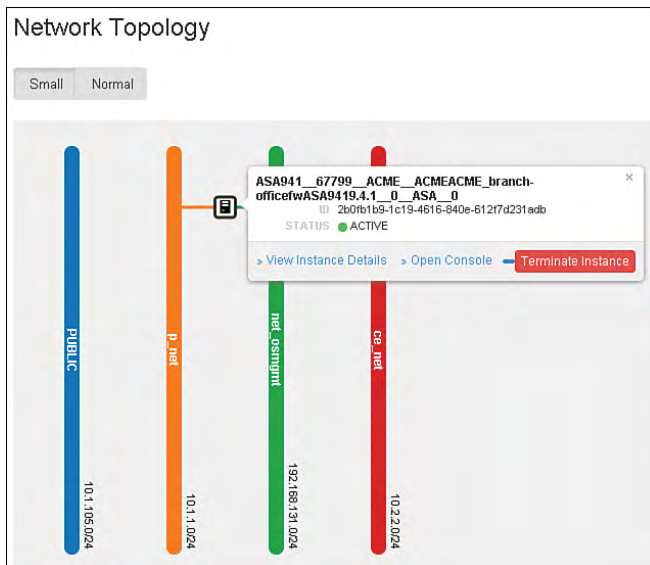


Figure 8-7 OpenStack Tenant:-ASA Service Chaining

- CSR (see Figure 8-8)

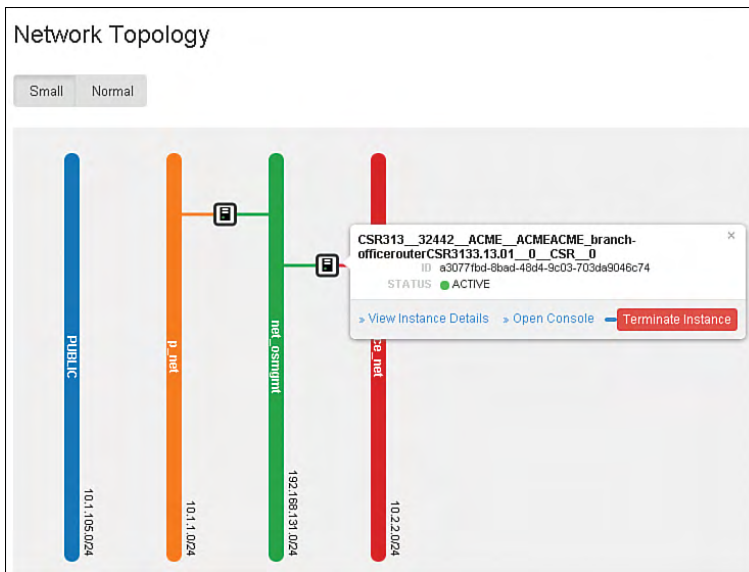


Figure 8-8 OpenStack Tenant: CSR Service Chaining

Orchestration

Orchestration leverages automation engines to drive a *user-defined* function across multiple environments. Some of these tools leverage automation engines like NSO. The following sections describe the tools that are commonly used for orchestration.

Virtual Managed Services (VMS)

VMS is a solution for a single orchestrator that provides automation of provisioning and management overview across the enterprise. Figure 8-9 shows a conceptual view from the workflow and service components across the enterprise.

Service workflows cover the service initiation for the user-defined workflows. This block consists of a tool that has a user-friendly interface to execute the workflows. The domain orchestration involves individual tools used for service initiation for different infrastructure solutions, as defined in Figure 8-9. This framework provides flexibility in terms of choosing a tool that is relevant to the solution for a specific user interface, domain orchestrator, or type of device configuration service.

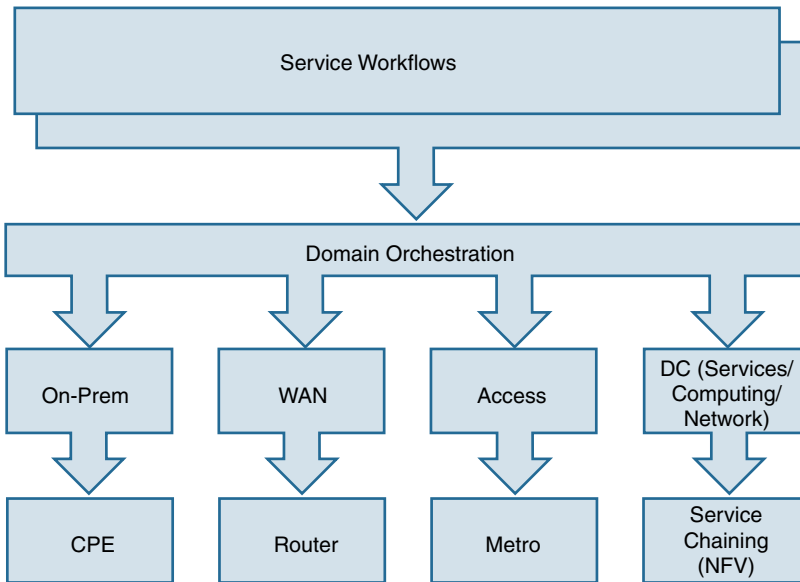


Figure 8-9 VMS Service Orchestration

As shown in Figure 8-10, the VMS solution covers the tools used for the tenant portal in the Cisco Prime catalog: Orchestration is covered by NSO, and the instantiation of the NFV elements is handled by the Elastic Services Controller (ESC). The NSO signals the ESC to instantiate a new NFV element. The ESC also stores the initial configuration and has a rule engine to access the elastic scale. This elastic scale is based on NFV monitoring via SNMP and custom scripts. This function helps create a new NFV element on demand if the CPU or load threshold increases in the existing NFV element. This is why the VMS solution uses the ESC controller instead of using NSO directly for instantiation.

The architectural framework followed by the VMS orchestration tool is in alignment with NFV Management and Orchestration (NFV MANO). NFV MANO covers the complete orchestration solution in three parts:

- **NFV Orchestrator**—Is responsible for provisioning network services and virtual devices. In VMS terminology, the NSO takes the role of the NFV Orchestrator.
- **VNF Manager**—Takes care of life cycle management of VNF instances. ESC in the VMS framework takes care of VNF manager functionality.
- **Virtualized Infrastructure Manager (VIM)**—Controls and manages the computing, storage, and network resources. VIM is equivalent to OpenStack’s vCenter.

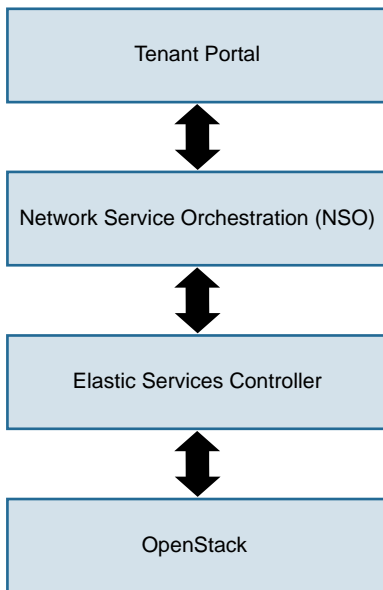


Figure 8-10 *VMS Architecture*

Cisco Prime Network Services Controller (PNSC)

PNSC is leveraged in enterprise private cloud deployments with traditional applications. PNSC uses an information model architecture, where each managed device is represented as a subcomponent (or object). A service profile is a collection of device policies and configuration templates predefined and applied to an NFV element or a group of NFV elements. These predefined service profiles can be applied across multiple tenants to take care of drafted service profiles using multiple NFV elements. This concept of using multiple NFV profiles to create a service for a tenant is called *service chaining*. Figure 8-11 provides a pictorial view of service chaining.

The NFV components are grouped together with a specific profile. The service chaining A and B profiles are used in tenants A and B. The same service chaining A profile can be used in tenant C (a new tenant). Creating these groups helps provision a tenant within minutes for a cloud environment. Cisco Prime has multi-hypervisor support and creates service chaining through NFV components in different hypervisors. Most of the enterprise environments have vCenter deployed in their data centers to facilitate computing virtualization. This tool fits in those environments to facilitate not only CSR provisioning and configuration but also other NFV elements, like ASA 1000V and VSG. The deployment of configuration for these NFV instances is GUI driven and largely used for traditional data center management and operations. These are some of the important features of the PNSC management solution:

- In a vCenter environment, PNSC associates with the hypervisor and can view all the host VMs seen by vCenter.

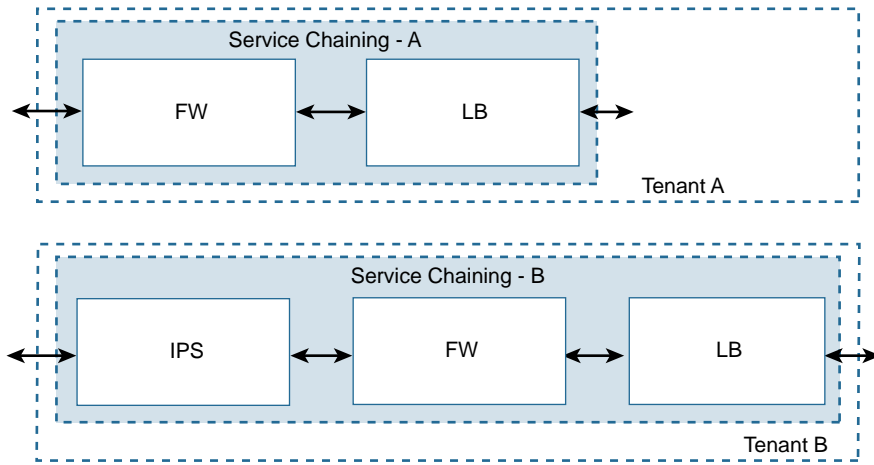


Figure 8-11 *Service Chaining Concept*

- To accelerate the performance of service chaining in the virtual environment, you can take advantage of the vPATH feature. The Cisco Nexus 1000V must be enabled for the vPATH. PNSC associates itself with the Nexus 1000V to get visibility for the administrator to manage all components that are important for service chaining.
- Configuration of all features for the NFV components can be done via the PNSC, such as security and routing policies.
- An administrator can manage, create, define, and monitor the NFV elements via this tool.
- A server admin can associate a VM with network services aligned to a proposed VM. This also can be done at VMware's vCenter to provide transparent mapping of network services on the computing orchestration tool.

Note The vPATH function of the Cisco Application Virtual Switch (AVS) or Nexus 1000V provides abstraction of the forwarding plane by redirecting packets to appropriate virtual service nodes that offer NFV services. vPATH steers traffic to optimize the traffic flow between the host and the service chained elements.

These are the main steps to remember for installing PNSC in a vCenter computing environment:

1. Create a tenant from vCenter and boot it with the correct OVA file. During this creation, the IP address for management should be added.
2. Register PNSC as a plugin to vCenter and accept the new plugin in vCenter. After the registration, create VM membership at PNSC, pointing to the vCenter's management IP address.
3. Create service chaining for a workflow with the required NFV element.

CSR 1000V Troubleshooting

This section describes common debugging and troubleshooting issues with the CSR 1000V platform. It also details the troubleshooting steps to take with ESXi as the hypervisor. Since the CSR 1000V is a virtual router, this section has been divided into three blocks involved in troubleshooting a CSR 1000V:

- **Architecture Overview**—This section provides a summary of the architecture that is covered in the earlier chapters and forms the base of the troubleshooting logic.
- **I/O Configuration**—This section gives an overview of the I/O models that are currently supported with the CSR 1000V virtual machine running on an ESXi hypervisor. It is important for a troubleshooter to understand the deployed I/O model prior to troubleshooting.
- **Debugging Packet Loss**—This section covers debugging techniques at the host/hypervisor/XE level and provides the technical knobs to collect the data at different architectural blocks covered while troubleshooting CSR 1000V.

Architecture Overview

Architecture of the CSR 1000V is covered in detail in the earlier chapters. Figure 8-12 illustrates the different components we analyze subsequently in the chapter to assist in debugging and troubleshooting packet flow issues.

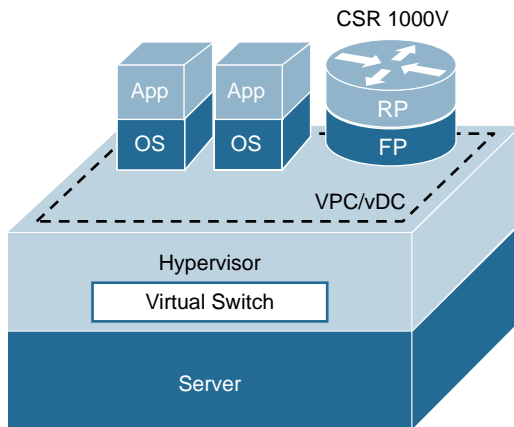


Figure 8-12 *CSR 1000V Virtual Router Architectural Overview*

From a high level, there are three components in the architecture (covered in detail in earlier chapters) of the CSR 1000V virtual router:

- **The CSR 1000V VM**—As detailed in previous chapters, the CSR VM leverages the IOS XE software infrastructure. It runs on a Linux kernel with applications running in the user space. IOSd and other IOS XE processes run in the user space.

The CSR 1000V control plane is made up of IOSd and other IOS XE infrastructure processes:

- The CSR data plane consists of the RX, PPE, and the HQF threads. All these components run on a single process within the QFP process, with multiple PPE threads. The software infrastructure of the CSR data plane ensures efficient movement of packets between the vNIC interface and the user space. The data path “punts” packets to the control plane if required, and the control plane “injects” data path packets to the vNIC interfaces.
- The CSR 1000V supports the following configuration:
 - 1-8 vCPUs, 2.5-16GB RAM, VMxNET3 drivers
- The IOS XE control plane and data plane share a single vCPU (in the case of a 1vCPU configuration). In the case of a multi-vCPU configuration, the control plane gets one vCPU. The remaining CPUs are allocated to the data plane.
- Packet drops and troubleshooting should first be done at the virtual machine level, using the IOS XE command-line interface that is provided.
- **The hypervisor software**—This troubleshooting section assumes ESXi from VMware as the hypervisor. As discussed in previous chapters, ESXi is a type 1 hypervisor that runs on the host machine and schedules resources. ESXi enables a virtualized environment by scheduling resources for guest operating systems that run on it. This chapter covers CSR 1000V on an ESXi hypervisor. It is important to understand troubleshooting at the hypervisor level as the hypervisor software in certain cases could be the reason for packet drops.
- **The host machine**—The host machine is the piece of hardware that the hypervisor manages.

I/O Configuration

On a physical router, the physical interfaces are either built in to the hardware or made available to the software by means of interface devices such as shared port adapters (SPA). In a virtualized environment, the VM uses the physical NIC on the host machine as an interface. There are multiple ways to connect a physical NIC to a CSR VM. It is important to understand this connection methodology to be able to successfully debug and troubleshoot an event. Performance and latency of a CSR 1000V will depend on how the I/O connection is set up.

The most common way a physical NIC is made available to the CSR VM is via a virtual switch.

vSwitch

A virtualized switch sits between a VM and the vNIC that the hypervisor presents. ESXi supports VMware’s distributed vSwitch and Cisco’s Nexus 1000V. This chapter covers VMware’s vSwitch.

Figure 8-13 illustrates the logical I/O model of a vSwitch.

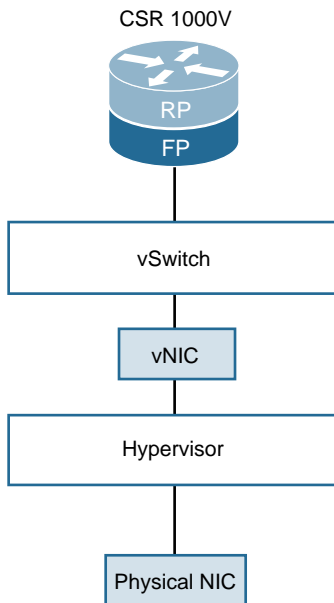


Figure 8-13 *vSwitch Logical I/O Model*

The virtual switch offers a Layer 2 connection between the VMs connected to it. It does not support physical switch features like IGMP snooping and Spanning Tree Protocol (STP).

Understanding the differences between a virtual switch and a physical switch will help you better debug issues. These are two important differences that need to be highlighted:

- There is no need for a virtual switch to learn unicast IP addresses or do IGMP snooping to enable them to learn multicast groups. The reason is that the ESX server has authoritative knowledge of the attached vNICs.
- Virtual switches do not need to support STP. The reason is because you do not need STP here to avoid loops. Because it is not possible to connect two virtual switches together, the only way to create loops in a virtualized switch would be to run bridging software as a guest VM. Hence STP is not required for a virtualized switch. The vSwitch should function in promiscuous mode. In this mode, only the objects defined within that port group have the option of receiving all incoming traffic on the vSwitch. Interfaces outside the port group in the vSwitch do not receive the traffic.

PCI Passthrough

In PCI passthrough mode, the hypervisor software that manages the physical NIC is completely bypassed using a Peripheral Component Interconnect Express (PCI-e) passthrough configuration; the guest VM gets direct access to the hardware NIC. This increases the throughput and reduces latency that would be introduced due to the hypervisor scheduling the NIC. The disadvantage here is that the NIC is dedicated to the VM, and virtualization for that NIC is not possible. You cannot use virtualization features (like vMotion) that the hypervisor has to offer but need to use the NICs that are supported by the guest VM. No driver is required on the ESX host machine. The guest VM needs to have a driver to support the NIC. This I/O model is also referred to as VM-DirectPath I/O.

CSR 1000V can work with Intel's 1G and 10G NICs in a PCI-e passthrough mode, as shown in Figure 8-14.

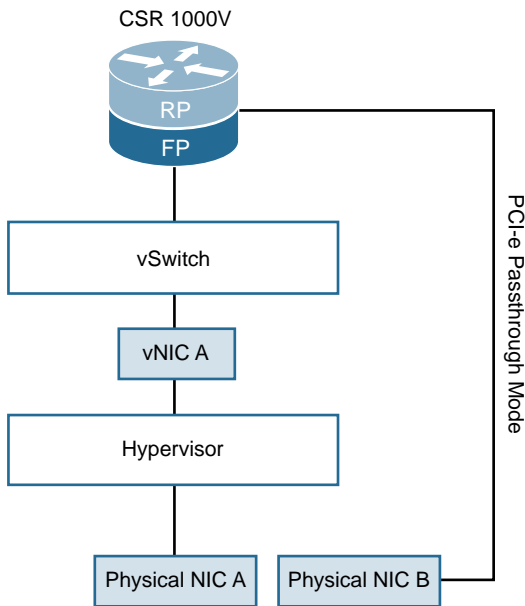


Figure 8-14 *PCI-e Passthrough Logical I/O Model*

SR-IOV (Single Root I/O Virtualization)

SR-IOV is a virtualization technique that allows a single PCI-e device to appear as multiple devices. This virtualization technology brings in the idea of physical and virtual functions (PFs and VFs).

A PF is a full-featured PCI-e function, which means it is managed like any other PCI-e device. A PF has a complete configuration resource, which means it completely owns the PCI-e device and can move data in and out of the device.

VFs work like PFs except for the configuration resource piece. A VF does not have a configuration resource because you do not want a VF to change the configuration. You just need the VF to move data in and out. The control for configuration change rests solely with the PFs, and the VF config should be dictated by the underlying PFs. VFs are not complete PCI-e devices, and so the hypervisor must be aware of the fact that it is dealing with an incomplete PCI-e device. This virtualization feature requires software support at the hypervisor level. For SR-IOV to work, you need BIOS and hardware support as well as support in software at the hypervisor/OS level.

Figure 8-15 illustrates the SR-IOV model.

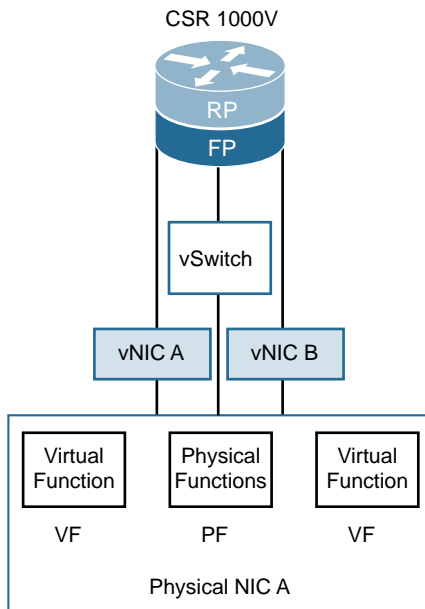


Figure 8-15 *SR-IOV Logical I/O Model*

Host Configurations

The hardware on which the CSR runs plays an important role in the throughput performance of the VM. The user must be aware of the hardware configurations of the host when debugging and troubleshooting packet flow and other issues with respect to the CSR 1000V. Following are some hardware specifications to look out for:

- **CPU**—Awareness on what kind of CPU the host machine has is significant because CPU architectures and throughputs change from hardware vendor to hardware vendor. This has a direct impact on the performance of the VM running on the host machine.
- **Sockets**—Most modern servers have CPUs with multiple sockets. When running an application over multiple sockets, the user sees degraded performance due to context switching, inefficient memory and cache access, and increased bus utilization.

- **BIOS**—BIOS settings on the hardware can impact the performance of the CSR. Following are certain BIOS settings the user should be aware of:
 - **Hyperthreading**—This technique makes a single physical core appear as two logical cores to the OS. The physical resources in the core are shared, but the system has additional overhead related to maintaining the states across the two logical cores. By default, hyperthreading is enabled to better utilize physical cores. However, for better performance, hyperthreading should be disabled.
 - **SpeedStep**—This setting on Intel processors is designed to reduce power consumption of the processor by changing the processor clock speed. It slows down the processor when it reckons that the CPU load is down. This gives unpredictable performance numbers for the CSR 1000V. The user must make sure this is disabled in BIOS to get consistent performance data.
 - **Power**—This setting may put the processor to sleep if the system is deemed idle. This is done to save power, and it impacts the CSR’s performance. Therefore, C-state should be set to C-0 to disable this feature in BIOS.

Debugging Packet Loss

The following sections detail the approach that needs to be taken to debug a packet loss on a CSR. Whenever you debug a packet loss on a CSR VM, it is important to keep in mind how the CSR VM is layered over the host machine and hypervisor. The following sections use ESXi as an example.

High-Level Packet Flow

As first mentioned in Chapter 4, “CSR 1000V Software Architecture,” there are three major data plane components:

- Rx thread
- Tx thread
- HQF (Hierarchical Queuing Framework) thread

All these components run in a single process under the QFP process umbrella. Multiple PPE threads serve requests within this QFP process.

In Figure 8-16, physical NICs are connected to two separate virtual switches. The packet comes in on the physical NIC. The hypervisor maps the packet to the vNIC and then forwards it to the attached vSwitch. The vSwitch forwards the frame based on the destination MAC address. The frame is then received by the CSR 1000V’s virtual interface. The packet then goes to the IOS XE code for processing, as detailed in Chapter 4.

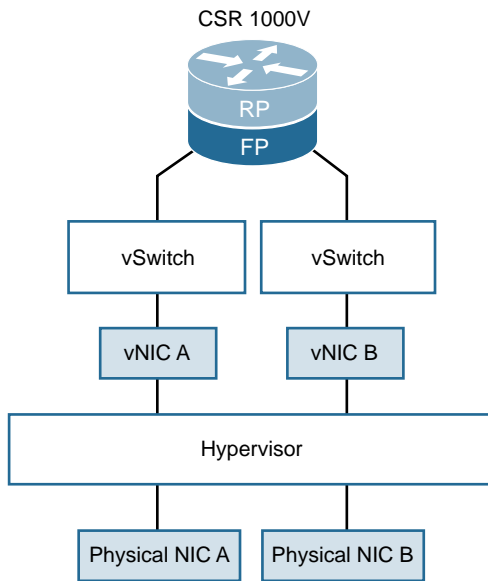


Figure 8-16 CSR Packet Flow with VMware ESXi

The following sections discuss certain things you need to check if you see packet loss on a CSR 1000V VM.

Appropriate Throughput License

As described in earlier chapters, CSR 1000V throughput is enforced by a software license. If you see packet drops, the first things to check are whether you have the right license and whether you are sending traffic more than the license is designed to police.

Example 8-5 shows how to check the throughput level and license details.

Example 8-5 How to Check Throughput Levels and License Details

```

Router# show platform hardware throughput level
The current throughput level is 100 kb/s

Router# show license detail
Index: 1 Feature: sec_100M Version: 1.0
License Type: Permanent
License State: Active, In Use
License Count: Non-Counted
License Priority: Medium
Store Index: 0
Store Name: Primary License Storage
Router#show platform hardware throughput level
  
```

```

The current throughput level is 5000000 kb/s

Router# show license detail
Index: 1          Feature: appx_5G          Version: 1.0
    License Type: Permanent
    License State: Active, In Use
    License Count: Non-Counted
    License Priority: Medium
    Store Index: 1
    Store Name: Primary License Storage
Index: 2          Feature: internal_service  Version: 1.0
    License Type: Paid Subscription
    Start Date:      N/A, End Date: May 25 2018
    License State: Active, Not in Use
    License Count: Non-Counted
    License Priority: Medium
    Store Index: 0
    Store Name: Primary License Storage

```

For the license to be enforced properly, its state should be Active, In Use. It is important to note that the CSR license policer is an aggregate policer and not an interface-level policer. If the license is for 1G, you can send 1G traffic combined through all interfaces. This should be the sum of all traffic going in or out of all interfaces from a CSR virtual machine.

To check whether traffic drops are due to traffic exceeding the license, use the following command and look for tail drops in the QFP subsystem:

```

Router# show platform hardware qfp active statistics drop clear | exc _0_
-----
Global Drop Stats          Packets          Octets
-----
TailDrop                   2018258         256333010

```

The license uses a policer that tail drops packets when you exceed the bandwidth enforced through the license.

Hardware and Software Speed Configurations

The CSR 1000V interfaces are by default 1G interfaces. To change the speeds of the interface, use the `speed` command in interface configuration mode, as shown in Example 8-6. If you are using 10G NIC hardware, you need to be aware that the interface still appears as a GigabitEthernet interface on the CSR. To check the speed, use the `show interface` command.

Example 8-6 *Changing the Speed of the Interface*

```

Router# sh run interface gigabitEthernet 1
Building configuration...
interface GigabitEthernet1
 ip address 10.201.131.1 255.255.255.0
 speed 10000
 no negotiation auto
end

Router# show interfaces gigabitEthernet 1
GigabitEthernet1 is up, line protocol is up
  Hardware is CSR vNIC, address is 000c.291a.7bd8 (bia 000c.291a.7bd8)
  Internet address is 10.201.131.1/24
  MTU 1500 bytes, BW 10000000 Kbit/sec, DLY 10 usec,
    reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full Duplex, 10000Mbps, link type is force-up, media type is RJ45
  output flow-control is unsupported, input flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input never, output 00:00:49, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
    4 packets input, 240 bytes, 0 no buffer
  Received 0 broadcasts (0 IP multicasts)
    0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog, 0 multicast, 0 pause input
    1 packets output, 60 bytes, 0 underruns
    0 output errors, 0 collisions, 1 interface resets
    0 unknown protocol drops
    0 babbles, 0 late collision, 0 deferred
    1 lost carrier, 0 no carrier, 0 pause output
    0 output buffer failures, 0 output buffers swapped out

```

The user needs to make sure the interface NIC is not oversubscribed, which means the user should not try to use the `speed` command when the physical NIC is a 1G module.

Example 8-7 is sample output from the `show interface` command on a CSR VM.

Example 8-7 *Interface show Command*

```

CSR1000v# show interfaces
GigabitEthernet1 is up, line protocol is up
  Hardware is CSR vNIC, address is 0050.5693.b409 (bia 0050.5693.b409)
  Internet address is 172.16.0.10/24
  MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
    reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full Duplex, 1000Mbps, link type is auto, media type is RJ45
  output flow-control is unsupported, input flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:00, output 00:05:48, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
    888185 packets input, 83771057 bytes, 0 no buffer
    Received 0 broadcasts (0 IP multicasts)
    0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog, 0 multicast, 0 pause input
    397945 packets output, 50738335 bytes, 0 underruns
    0 output errors, 0 collisions, 0 interface resets
    66567 unknown protocol drops
    0 babbles, 0 late collision, 0 deferred
    0 lost carrier, 0 no carrier, 0 pause output
    0 output buffer failures, 0 output buffers swapped out

```

L2 MTU

The CSR 1000V supports an L2 MTU of 1500–9216. This, as in any other IOS/IOS XE device, can be changed using the `mtu` command in interface configuration mode. It is, however, important to note that just changing the MTU on the CSR does not guarantee a configured MTU. The user needs to make sure the vSwitch is configured with an MTU value that equals the MTU configured on the CSR interface.

Interface-to-NIC Mapping

To be sure of what interface on the CSR VM matches the interface configured on ESXi, the user should run the following command on the CSR VM:

```
CSR1000v# show platform software vnic-if interface-mapping
```

| Interface Name | Driver Name | Mac Addr |
|------------------|-------------|----------------|
| GigabitEthernet3 | vmxnet3 | 0050.5693.c25f |
| GigabitEthernet2 | vmxnet3 | 0050.5693.3208 |
| GigabitEthernet1 | vmxnet3 | 0050.5693.b409 |

The user should note the MAC address and verify it with the interface configuration on the ESXi host.

Figure 8-17 illustrates the MAC address-to-interface mappings.

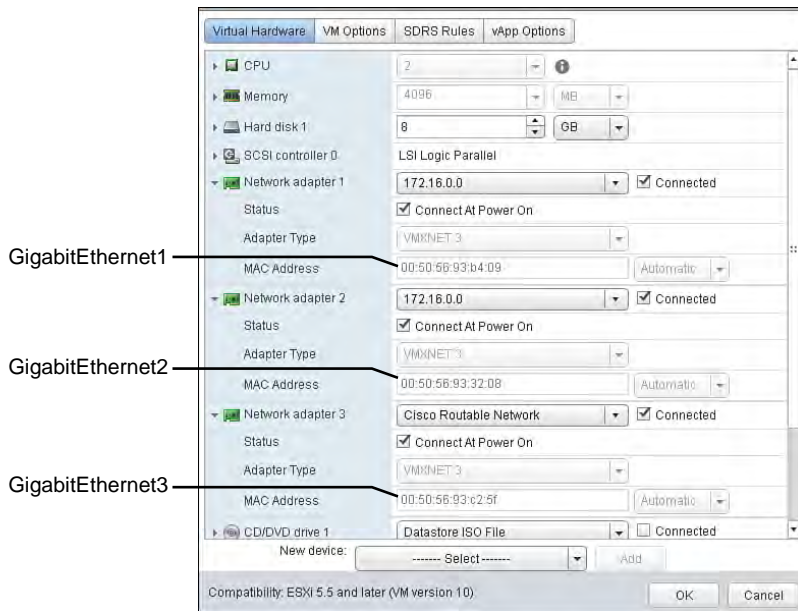


Figure 8-17 ESXi with Mapping

CPU and Memory Usage

With packet loss, it is always a good idea to check the overall health of the CSR. Example 8-8 shows the IOS XE command to find the health of the VM from IOS XE’s perspective.

Example 8-8 shows the control-processor CLI that gives the control and data plane memory and CPU status. IOS XE drops packets when the CPU utilization reaches 100%.

Example 8-8 *Control-Processor CLI Output*

```
Router# show platform software status control-processor brief
Load Average
  Slot  Status  1-Min  5-Min 15-Min
  RP0   Healthy  2.10  1.35  0.59

Memory (kB)
  Slot  Status  Total      Used (Pct)    Free (Pct)    Committed (Pct)
  RP0   Healthy  8117052    3226704 (40%)  4890348 (60%)  3711452 (46%)

CPU Utilization
  Slot  CPU    User   System   Nice   Idle   IRQ   SIRQ  IOWait
  RP0   0     1.93  0.90    0.00  96.36  0.00  0.80  0.00
       1     95.01 4.98    0.00  0.00  0.00  0.00  0.00
```

Figure 8-18 illustrates high-level packet flow.

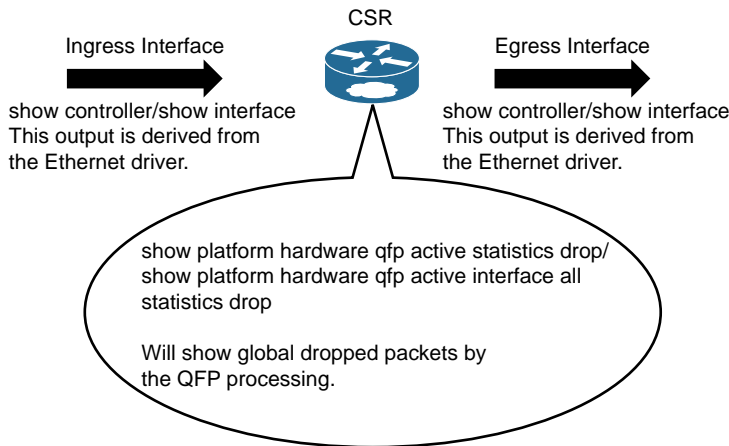


Figure 8-18 *CSR Packet Flow: High Level*

Make sure a packet is received and transmitted by the vNIC. The first step is to make sure the packets are properly received and transmitted by the CSR vNIC interface. The IOS `show interfaces` command displays the packet statistics for a vNIC interface.

Example 8-9 shows the CLI output for the `show interfaces` command.

Example 8-9 show interfaces *CLI Output*

```

CSR1000v# show interfaces Gig1 | begin 5 min
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
    888185 packets input, 83771057 bytes, 0 no buffer
    Received 0 broadcasts (0 IP multicasts)
    0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog, 0 multicast, 0 pause input
    397945 packets output, 50738335 bytes, 0 underruns
    0 output errors, 0 collisions, 0 interface resets
    66567 unknown protocol drops
    0 babbles, 0 late collision, 0 deferred
    0 lost carrier, 0 no carrier, 0 pause output
    0 output buffer failures, 0 output buffers swapped out

```

To check the vNIC driver statistics, use the `show interface controller` command. The CSR vNIC driver should be fast enough to get packets onto the data plane. This driver should not be a bottleneck because the packet drops at the driver level are random. You can increase the vNIC queue size if there are drops at the driver level.

Example 8-10 shows the `show interface controller` output. The key part of the CLI output that needs to be examined during troubleshooting is highlighted.

Example 8-10 *Interface Controller Output*

```

CSR1000v# show interface gigabitEthernet 1 controller
GigabitEthernet1 is up, line protocol is up
Hardware is CSR vNIC, address is 0050.5693.b409 (bia 0050.5693.b409)
Internet address is 172.16.0.10/24
MTU 1500 bytes, BW 1000000 Kbit/sec, DLY 10 usec,
    reliability 255/255, txload 1/255, rxload 1/255
Encapsulation ARPA, loopback not set
Keepalive set (10 sec)
Full Duplex, 1000Mbps, link type is auto, media type is RJ45
output flow-control is unsupported, input flow-control is unsupported
ARP type: ARPA, ARP Timeout 04:00:00
Last input 00:00:00, output 00:06:56, output hang never
Last clearing of "show interface" counters never
Input queue: 0/375/0/0 (size/max/drops/flushes); Total output drops: 0
Queueing strategy: fifo
Output queue: 0/40 (size/max)
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
    888241 packets input, 83775638 bytes, 10 no buffer

```

```

Received 0 broadcasts (0 IP multicasts)
0 runs, 0 giants, 0 throttles
72 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
0 watchdog, 0 multicast, 0 pause input
397961 packets output, 50739519 bytes, 0 underruns
0 output errors, 0 collisions, 0 interface resets
66570 unknown protocol drops
0 babbles, 0 late collision, 0 deferred
0 lost carrier, 0 no carrier, 0 pause output
0 output buffer failures, 0 output buffers swapped out
GigabitEthernet1 - Gi1 is mapped to eth0 on VXE
DPIF Rx Drop 76456 Packets 888242
Driver Rx Stops 0 DPIF Rx Congestion Drop 88
Detailed interface statistics:
TSO pkts tx 0
TSO bytes tx 0
ucast pkts tx 237307
ucast bytes tx 38680975
mcast pkts tx 160633
mcast bytes tx 12056879
bcast pkts tx 21
bcast bytes tx 1665
pkts tx err 0
pkts tx discard 0
drv dropped tx total 0
  too many frags 0
  giant hdr 0
  hdr err 0
  tso 0
ring full 0
pkts linearized 0
hdr cloned 0
giant hdr 0
LRO pkts rx 0
LRO byte rx 0
ucast pkts rx 131864
ucast bytes rx 13300931
mcast pkts rx 693292
mcast bytes rx 65646631
bcast pkts rx 63086
bcast bytes rx 4828136
pkts rx out of buf 0
pkts rx err 0
drv dropped rx total 0
  err 0

```



```

fcs 0
rx buf alloc fail 0
tx timeout count 0    too many frags 0
giant hdr 0
hdr err 0

```

The errors in Example 8-10 indicate a completely overloaded CSR data plane.

The packets make it to the CSR data plane. The `show interfaces <name> stats` command tells the user whether the data plane received packets from the vNIC interface:

```

CSR1000v# show interfaces GigabitEthernet 1 stats
GigabitEthernet1
  Switching path   Pkts In   Chars In   Pkts Out   Chars Out
  Processor        553496   46665118   5897       378453
  Route cache      0         0          0          0
Distributed cache  888252   83777385   397963     50739667
  Total            1441748  130442503  403860     51118120

```

To check the QFP subsystem (data plane) drops, use the `statistics drop` command:

```

CSR1000v# show platform hardware qfp active statistics drop
-----
Global Drop Stats                Packets                Octets
-----
Disabled                          410349859              24621130428
Ipv4NoAdj                          215639                  16345664
Ipv4NoRoute                        10213456                755734442
TailDrop                            10                       660

```

This is an aggregate of statistics and displays all drops on the QFP.

Table 8-1 explains some of the common drops.

Table 8-1 IOS XE Drop Types

| Drop Type | Possible Reason |
|---------------|---|
| BadUIdbSubIdx | This means that a received packet could not be mapped to a known or configured interface. This could be because of a race condition where a packet could be received before the configuration is applied to the data plane. |
| BqsOor | Out-of-packet memory causes the drops, and this happens due to congestion at the BQS. |
| Disabled | This error means that packets are being received on an interface that is not in up state. Verify this by using <code>show ip interface brief</code> and ensuring that the interface status is UP. |

| Drop Type | Possible Reason |
|---------------------|--|
| Ipv4NoAdj | This means ARP is not resolved on that interface. |
| Ipv4NoRoute | This error indicates that the destination IP address is not present in the routing table. Check if the IP address assigned to the interface is correct. |
| TailDrop | This error is generally due to packets getting dropped at the egress interface. One of the common reasons for this error is sending traffic beyond license. Another reason can be overload by packet fragmentation. This can happen when packets are fragmented due to the MTU size. |
| UnconfiguredIpv4Fia | This means the IP packet is being received and the IPv4 address is not set on that interface. |
| UnconfiguredIpv6Fia | This means the IP packet is being received and the IPv6 address is not set on that interface. |

IOS XE allows you to get into the details of the feature drops, as shown in Example 8-11, which shows the debugging options at the QFP level.

Example 8-11 QFP Feature Debugging Options

```

CSR1000v# show platform hard qfp active feature ?
 acl          Access Control List
 aic          QFP AIC information
 alg          QFP ALG
 aps          QFP APS information
 bfd          QFP BFD
 bridge-domain QFP Bridge domain feature Information
 cef-mpls     Show cef mpls
 cidb        QFP CIDB
 conf-sw      Software Conference
 cts          QFP CTS information
 cws          QFP CWS information
 cxsc        QFP CXSC feature Information
 docsis      QFP DOCSIS information
 dpss        ONE-P Datapath Service Set feature Information
 ecfm        QFP ECFM information
 epbr        Enhanced Policy Based Routing feature Information
 epc         QFP Embedded Packets capture Feature Information
 erspan      QFP Encapsulated Remote Switch Port Analyzer information
 evc         QFP EVC information
 evtmon      QFP EVENT monitor information
 fhs         QFP First Hop Security feature Information
 firewall    QFP Firewall information

```

```

fmd          QFP Flow-Metadata feature Information
fme          QFP Flow Metric Engine feature Information
fnf          QFP NetFlow
frame-relay  frame relay dp information
icmp         QFP ICMP information
icmpv6       QFP ICMPV6 information
ipfrag       QFP FRAGMENTATION
iphc         QFP IPHC
ipsec        QFP IPSEC
l2bd         QFP L2BD
l2es         QFP L2ES
l2mc         Show L2 multicast route information
l2vpn        layer2 VPN information
lisp         QFP LISP information
mlppp        QFP MLPPP information
mma          QFP Metric-Mediation-Agent feature Information
multicast    Show multicast route information
nat          QFP NAT information
nat64        QFP NAT64 information
nbar         QFP NBAR information
otv          QFP OTV information
packet-trace Packet-Trace information
pbr          QFP PBR information
pfr          QFP Performance Routing (Pfr) Information
pfrv3        Connected Enterprise feature Information
qos          QoS information
sbc          Session Border Controller
service-wire service wire configuration
smi          QFP Secure Management Interface information
sslvpn       QFP SSLVPN information
subscriber   QFP Subscriber (ESS) information
tcp          QFP TCP information
td           QFP TD
tunnel       QFP Tunnel
utd          QFP UTD feature Information
vpls         QFP VPLS information
wccp         QFP wccp services information

```

Feature specific drops

```
CSR1000v# show platform hard qfp active feature ipsec datapath drop all | incl SPI
```

```

-----
Drop Type  Name                                     Packets
-----
          4  IN_US_V4_PKT_SA_NOT_FOUND_SPI           2322
          7  IN_TRANS_V4_IPSEC_PKT_NOT_FOUND_SPI      0
         12  IN_US_V6_PKT_SA_NOT_FOUND_SPI            0

```

To check the QFP process utilization, use the following command:

```
CSR1000v# show platform hardware qfp active datapath utilization summary
```

| CPP 0: | | 5 secs | 1 min | 5 min | 60 min |
|------------------|-------------|--------|-------|-------|--------|
| Input: | Total (pps) | 12 | 12 | 12 | 10 |
| | (bps) | 16024 | 12760 | 12408 | 11360 |
| Output: | Total (pps) | 4 | 4 | 5 | 3 |
| | (bps) | 12664 | 10032 | 10840 | 8520 |
| Processing: Load | (pct) | 0 | 0 | 0 | 0 |

The QFP starts dropping packets if the utilization reaches 100%. Pushing in more CPUs should solve this problem. A high CPU number here indicates that the IOS XE software needs more QFP processing power.

To check the packet drops on the RX process of the data plane, use the following command shown in Example 8-12, which provides the data path `sw-nic` information.

Example 8-12 `sw-nic` CLI Output

```
CSR1000v# show platform hardware qfp active datapath infrastructure sw-nic
PMAP info:
  poll err 0; epochs: pmap 0 wait_all 0
  poll calls 0

ZNM info:
  poll err 0; epochs: znm 3 wait_all 3
  poll calls 28079227 priorities 1
  im-tx - active

znm 8d689040 device Gi1 (GigabitEthernet1)
Rx: pkts 888361 bytes 83786013 xoff
  Ring read 881774 empty 0 rx_avail 0
  revents 0 len err 0 credit err 0 audit 0
  im-alloc err 0
Tx: pkts 397992 bytes 50741813 send 0 forced-txsync 320438
  fill 0 poll 0 thd_poll 0
  full 0 lowater 0 hiwater 0
  avail 2046 batch 397992 tx_batch_sz 0 sendnow 0 im_alloc_err 0

znm 8d698040 device Gi2 (GigabitEthernet2)
Rx: pkts 9 bytes 568 xoff
  Ring read 9 empty 0 rx_avail 0
  revents 0 len err 0 credit err 0 audit 0
  im-alloc err 0
Tx: pkts 0 bytes 0 send 0 forced-txsync 0
  fill 0 poll 0 thd_poll 0
  full 0 lowater 0 hiwater 0
```

```

avail 2047 batch 0 tx_batch_sz 0 sendnow 0 im_alloc_err 0
znm 8d6a5040 device Gi3 (GigabitEthernet3)
Rx: pkts 6237543 bytes 1078457641 xoff
Ring read 6203494 empty 0 rx_avail 0
revents 0 len err 0 credit err 0 audit 0
im-alloc err 0
Tx: pkts 12621 bytes 1303109 send 0 forced-txsync 11888
fill 0 poll 0 thd_poll 0
full 0 lowater 0 hiwater 0
avail 2046 batch 12621 tx_batch_sz 0 sendnow 0 im_alloc_err 0

```

vSwitch Packet Drops

A vSwitch is not like a physical switch in that the hypervisor (ESXi) programs a vSwitch with all the vNIC MAC addresses into the vSwitch's MAC address table. So a vSwitch is not a learning switch. It drops any packets for destination MAC addresses it is unaware of. It always expects to receive packets with MAC addresses it is aware of. If you have frames with MAC addresses that are not on the vNIC interfaces connected to the vSwitch, the vSwitch drops those frames. If you want a vSwitch to not drop frames with unknown MAC addresses, you have to configure the vNIC in promiscuous mode.

ESXi Packet Debugging

Example 8-13 shows some useful ESXi commands. The useful ESXi CLI can be used for packet flow troubleshooting of a CSR VM.

Example 8-13 VM List on an ESXi Host

```

# esxcli network vm list
World ID   Name                               Num Ports Networks
-----
9966      MS_AD_DNS                           1 VM Network
10151     Cisco_CSR_1000V_3.16.0              3 VM Network, Flat, SNAT
510934    vMS                                  2 VM Network, SNAT
# esxcli network vm port list -w <worldID>
# esxcli network vm port list -w 10151
Port ID: 33554443
vSwitch: vSwitch0
Portgroup: VM Network
DVPort ID:
MAC Address: 00:0c:29:1a:7b:ec
IP Address: 0.0.0.0
Team Uplink: vmnic0
Uplink Port ID: 33554434
Active Filters:

```

```

Port ID: 33554444
vSwitch: vSwitch0
Portgroup: Flat
DVPort ID:
MAC Address: 00:0c:29:1a:7b:e2
IP Address: 0.0.0.0
Team Uplink: vmnic0
Uplink Port ID: 33554434
Active Filters:

Port ID: 33554445
vSwitch: vSwitch0
Portgroup: SNAT
DVPort ID:
MAC Address: 00:0c:29:1a:7b:d8
IP Address: 0.0.0.0
Team Uplink: vmnic0
Uplink Port ID: 33554434
Active Filters:

```

Example 8-14 shows the vNIC stats from an ESXi host.

Example 8-14 *ESXi NIC Stats*

```

# esxcli network nic stats get -n vmnic0
NIC statistics for vmnic0
Packets received: 4537423
Packets sent: 2892949
Bytes received: 4072878617
Bytes sent: 323830118
Receive packets dropped: 0
Transmit packets dropped: 0
Total receive errors: 0
Receive length errors: 0
Receive over errors: 0
Receive CRC errors: 0
Receive frame errors: 0
Receive FIFO errors: 0
Receive missed errors: 0
Total transmit errors: 0
Transmit aborted errors: 0
Transmit carrier errors: 0
Transmit FIFO errors: 0
Transmit heartbeat errors: 0
Transmit window errors: 0   Transmit heartbeat errors: 0
Transmit window errors: 0

```

It is recommended that you increase the receive ring buffer size to 4K because the device driver is not fast enough to retrieve these packets and post new buffers. You do this by using the following command:

```
# ethtool -G <vmnic> rx <size>
```

If this does not help, check the version of the driver by using the following command:

```
# ethtool -i <vmnic>
```

In practice, it should look like this:

```
# ethtool -i vmnic0
driver: igb
version: 2.1.11.1
firmware-version: 1.5-9
bus-info: 0000:01:00.0
```

From the VMware support site, check whether an updated version of this driver is available. Sometimes upgrading the driver helps resolve issues.

These same details of packet statistics can also be obtained from vCenter, as shown in Figure 8-19. They are present at Hosts > Monitor > Performance > Advanced. Using the chart options, you can select the chart metrics as Network, object type as the VM's VM NIC, and other desired counters, such as data transmit rate, data receive rate, and packet transmit errors.

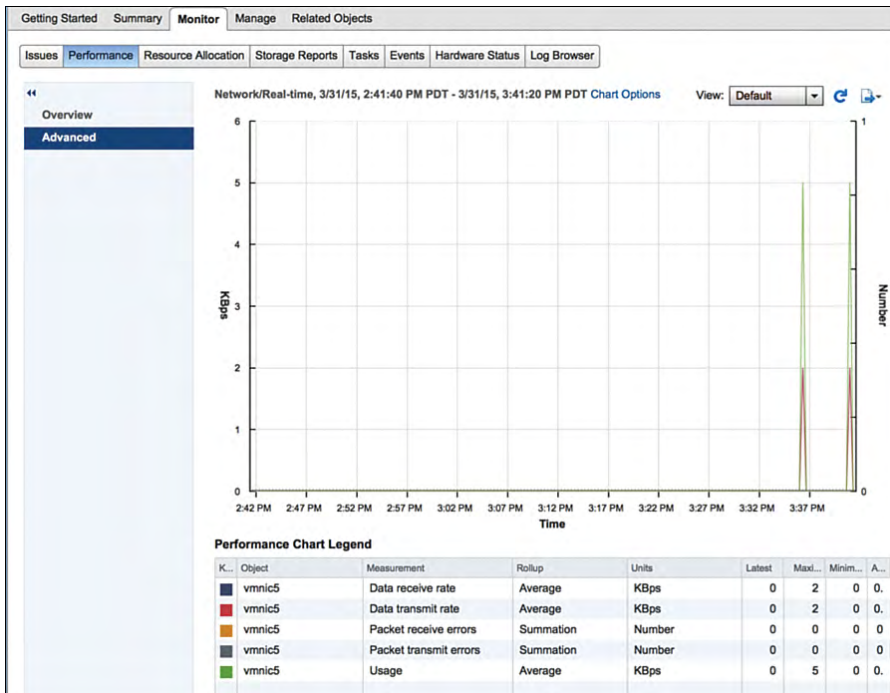


Figure 8-19 Network Statistics of NICs Using vCenter

You can check whether the packets are being received at vmnic to the vSwitch ingress port. Example 8-15 shows the port stats from an ESXi host.

Example 8-15 *ESXi Port Stats*

```
# esxcli network port stats get -p 33554443
Packet statistics for port 33554443
Packets received: 1837526
Packets sent: 3223
Bytes received: 173220390
Bytes sent: 241813
Broadcast packets received: 688401
Broadcast packets sent: 14
Multicast packets received: 1148372
Multicast packets sent: 2615
Unicast packets received: 753
Unicast packets sent: 594
Receive packets dropped: 1599982
Transmit packets dropped: 0
```

One of the reasons for the packet drops might be the high CPU Ready Time for the VM. The CPU Ready Time is the time when the VM is ready to run but the hypervisor is not able to schedule it on a physical processor. A high value indicates poor performance. This value can be found by running the following command on the host:

```
# esxtop
```

The value returned should ideally be zero. If it is not, there is not enough CPU for the VM to be scheduled in time, which may lead to packet drops. Reduce the number of VMs on the ESXi host if you run into this issue.

Summary

This chapter reviews three import topics: automation, orchestration, and troubleshooting. The first part of this chapter disambiguates the concepts of automation and orchestration, using the key tools available. After reading this chapter, you should be able to appreciate the difference between these two concepts and should be in a position to choose the correct tool for your virtualized infrastructure.

This chapter also describes how to troubleshoot a CSR VM on an ESXi hypervisor. You should be able to follow the troubleshooting approach described here and leverage the command-line outputs to get a better situational awareness of a CSR VM.

Sample Answer File for Packstack

The following output is the complete output of the abbreviated Example 7-1 from Chapter 7, “CSR in the SDN Framework.” It provides a sample answer file for Packstack.

```
[general]

# Path to a public key to install on servers. If a usable key has not
# been installed on the remote servers, the user is prompted for a
# password and this key is installed so the password will not be
# required again.
CONFIG_SSH_KEY=/root/.ssh/id_rsa.pub

# Default password to be used everywhere (overridden by passwords set
# for individual services or users).
CONFIG_DEFAULT_PASSWORD=lab

# Specify 'y' to install MariaDB.
CONFIG_MARIADB_INSTALL=y

# Specify 'y' to install OpenStack Image Service (glance).
CONFIG_GLANCE_INSTALL=y

# Specify 'y' to install OpenStack Block Storage (cinder).
CONFIG_CINDER_INSTALL=y

# Specify 'y' to install OpenStack Shared File System (manila).
CONFIG_MANILA_INSTALL=y

# Specify 'y' to install OpenStack Compute (nova).
CONFIG_NOVA_INSTALL=y
```

```
# Specify 'y' to install OpenStack Networking (neutron); otherwise,
# Compute Networking (nova) will be used.
CONFIG_NEUTRON_INSTALL=y

# Specify 'y' to install OpenStack Dashboard (horizon).
CONFIG_HORIZON_INSTALL=y

# Specify 'y' to install OpenStack Object Storage (swift).
CONFIG_SWIFT_INSTALL=y

# Specify 'y' to install OpenStack Metering (ceilometer).
CONFIG_CEILOMETER_INSTALL=y

# Specify 'y' to install OpenStack Orchestration (heat).
CONFIG_HEAT_INSTALL=y

# Specify 'y' to install OpenStack Data Processing (sahara).
CONFIG_SAHARA_INSTALL=n

# Specify 'y' to install OpenStack Database (trove).
CONFIG_TROVE_INSTALL=n

# Specify 'y' to install OpenStack Bare Metal Provisioning (ironic).
CONFIG_IRONIC_INSTALL=n

# Specify 'y' to install the OpenStack Client packages (command-line
# tools). An admin "rc" file will also be installed.
CONFIG_CLIENT_INSTALL=y

# Comma-separated list of NTP servers. Leave plain if Packstack
# should not install ntpd on instances.
CONFIG_NTP_SERVERS=

# Specify 'y' to install Nagios to monitor OpenStack hosts. Nagios
# provides additional tools for monitoring the OpenStack environment.
CONFIG_NAGIOS_INSTALL=y

# Comma-separated list of servers to be excluded from the
# installation. This is helpful if you are running Packstack a second
# time with the same answer file and do not want Packstack to
# overwrite these server's configurations. Leave empty if you do not
# need to exclude any servers.
EXCLUDE_SERVERS=

# Specify 'y' if you want to run OpenStack services in debug mode;
```

```
# otherwise, specify 'n'.
CONFIG_DEBUG_MODE=n

# IP address of the server on which to install OpenStack services
# specific to the controller role (for example, API servers or
# dashboard).
CONFIG_CONTROLLER_HOST=192.168.1.22

# List of IP addresses of the servers on which to install the Compute
# service.
CONFIG_COMPUTE_HOSTS=192.168.1.22

# List of IP addresses of the server on which to install the network
# service such as Compute networking (nova network) or OpenStack
# Networking (neutron).
CONFIG_NETWORK_HOSTS=192.168.1.22

# Specify 'y' if you want to use VMware vCenter as hypervisor and
# storage; otherwise, specify 'n'.
CONFIG_VMWARE_BACKEND=n

# Specify 'y' if you want to use unsupported parameters. This should
# be used only if you know what you are doing. Issues caused by using
# unsupported options will not be fixed before the next major release.
CONFIG_UNSUPPORTED=n

# Specify 'y' if you want to use subnet addresses (in CIDR format)
# instead of interface names in following options:
# CONFIG_NOVA_COMPUTE_PRIVIF, CONFIG_NOVA_NETWORK_PRIVIF,
# CONFIG_NOVA_NETWORK_PUBIF, CONFIG_NEUTRON_OVS_BRIDGE_IFACES,
# CONFIG_NEUTRON_LB_INTERFACE_MAPPINGS, CONFIG_NEUTRON_OVS_TUNNEL_IF.
# This is useful for cases when interface names are not same on all
# installation hosts.
CONFIG_USE_SUBNETS=n

# IP address of the VMware vCenter server.
CONFIG_VCENTER_HOST=

# User name for VMware vCenter server authentication.
CONFIG_VCENTER_USER=

# Password for VMware vCenter server authentication.
CONFIG_VCENTER_PASSWORD=

# Name of the VMware vCenter cluster.
```

```
CONFIG_VCENTER_CLUSTER_NAME=

# (Unsupported!) IP address of the server on which to install
# OpenStack services specific to storage servers such as Image or
# Block Storage services.
CONFIG_STORAGE_HOST=192.168.1.22

# (Unsupported!) IP address of the server on which to install
# OpenStack services specific to OpenStack Data Processing (sahara).
CONFIG_SAHARA_HOST=192.168.1.22

# Specify 'y' to enable the EPEL repository (Extra Packages for
# Enterprise Linux).
CONFIG_USE_EPEL=y

# Comma-separated list of URLs for any additional yum repositories,
# to use for installation.
CONFIG_REPO=

# Specify 'y' to enable the RDO testing repository.
CONFIG_ENABLE_RDO_TESTING=n

# To subscribe each server with Red Hat Subscription Manager, include
# this with CONFIG_RH_PW.
CONFIG_RH_USER=

# To subscribe each server to receive updates from a Satellite
# server, provide the URL of the Satellite server. You must also
# provide a user name (CONFIG_SATELLITE_USERNAME) and password
# (CONFIG_SATELLITE_PASSWORD) or an access key (CONFIG_SATELLITE_AKEY)
# for authentication.
CONFIG_SATELLITE_URL=

# To subscribe each server with Red Hat Subscription Manager, include
# this with CONFIG_RH_USER.
CONFIG_RH_PW=

# Specify 'y' to enable RHEL optional repositories.
CONFIG_RH_OPTIONAL=y

# HTTP proxy to use with Red Hat Subscription Manager.
CONFIG_RH_PROXY=

# Port to use for Red Hat Subscription Manager's HTTP proxy.
CONFIG_RH_PROXY_PORT=
```

```
# User name to use for Red Hat Subscription Manager's HTTP proxy.
CONFIG_RH_PROXY_USER=

# Password to use for Red Hat Subscription Manager's HTTP proxy.
CONFIG_RH_PROXY_PW=

# User name to authenticate with the RHN Satellite server; if you
# intend to use an access key for Satellite authentication, leave this
# blank.
CONFIG_SATELLITE_USER=

# Password to authenticate with the RHN Satellite server; if you
# intend to use an access key for Satellite authentication, leave this
# blank.
CONFIG_SATELLITE_PW=

# Access key for the Satellite server; if you intend to use a user
# name and password for Satellite authentication, leave this blank.
CONFIG_SATELLITE_AKEY=

# Certificate path or URL of the certificate authority to verify that
# the connection with the Satellite server is secure. If you are not
# using Satellite in your deployment, leave this blank.
CONFIG_SATELLITE_CACERT=

# Profile name that should be used as an identifier for the system in
# RHN Satellite (if required).
CONFIG_SATELLITE_PROFILE=

# Comma-separated list of flags passed to the rhnreg_ks command
# (novirtinfo, norhnsd, nopackages).
CONFIG_SATELLITE_FLAGS=

# HTTP proxy to use when connecting to the RHN Satellite server (if
# required).
CONFIG_SATELLITE_PROXY=

# User name to authenticate with the Satellite-server HTTP proxy.
CONFIG_SATELLITE_PROXY_USER=

# User password to authenticate with the Satellite-server HTTP proxy.
CONFIG_SATELLITE_PROXY_PW=

# Specify filepath for CA cert file. If CONFIG_SSL_CACERT_SELFSIGN is
# set to 'n' it has to be preexisting file.
```

```
CONFIG_SSL_CACERT_FILE=/etc/pki/tls/certs/selfcert.crt

# Specify filepath for CA cert key file. If
# CONFIG_SSL_CACERT_SELFSIGN is set to 'n' it has to be preexisting
# file.
CONFIG_SSL_CACERT_KEY_FILE=/etc/pki/tls/private/selfkey.key

# Enter the path to use to store generated SSL certificates in.
CONFIG_SSL_CERT_DIR=~/.packstackca/

# Specify 'y' if you want Packstack to pregenerate the CA
# Certificate.
CONFIG_SSL_CACERT_SELFSIGN=y

# Enter the selfsigned CAcert subject country.
CONFIG_SELFSIGN_CACERT_SUBJECT_C=US

# Enter the selfsigned CAcert subject state.
CONFIG_SELFSIGN_CACERT_SUBJECT_ST=VA

# Enter the selfsigned CAcert subject location.
CONFIG_SELFSIGN_CACERT_SUBJECT_L=Belmont

# Enter the selfsigned CAcert subject organization.
CONFIG_SELFSIGN_CACERT_SUBJECT_O=openstack

# Enter the selfsigned CAcert subject organizational unit.
CONFIG_SELFSIGN_CACERT_SUBJECT_OU=packstack

# Enter the selfsigned CAcert subject common name.
CONFIG_SELFSIGN_CACERT_SUBJECT_CN=openstack-csr.cisco.com

CONFIG_SELFSIGN_CACERT_SUBJECT_MAIL=admin@openstack-csr.cisco.com

# Service to be used as the AMQP broker (qpid, rabbitmq).
CONFIG_AMQP_BACKEND=rabbitmq

# IP address of the server on which to install the AMQP service.
CONFIG_AMQP_HOST=192.168.1.22

# Specify 'y' to enable SSL for the AMQP service.
CONFIG_AMQP_ENABLE_SSL=n

# Specify 'y' to enable authentication for the AMQP service.
CONFIG_AMQP_ENABLE_AUTH=n
```

```
# Password for the NSS certificate database of the AMQP service.
CONFIG_AMQP_NSS_CERTDB_PW=lab

# User for AMQP authentication.
CONFIG_AMQP_AUTH_USER=amqp_user

# Password for AMQP authentication.
CONFIG_AMQP_AUTH_PASSWORD=lab

# IP address of the server on which to install MariaDB. If a MariaDB
# installation was not specified in CONFIG_MARIADB_INSTALL, specify
# the IP address of an existing database server (a MariaDB cluster can
# also be specified).
CONFIG_MARIADB_HOST=192.168.1.22

# User name for the MariaDB administrative user.
CONFIG_MARIADB_USER=root

# Password for the MariaDB administrative user.
CONFIG_MARIADB_PW=16c20883fa11493d

# Password to use for the Identity service (keystone) to access the
# database.
CONFIG_KEYSTONE_DB_PW=0446a5983fb64729

# Default region name to use when creating tenants in the Identity
# service.
CONFIG_KEYSTONE_REGION=RegionOne

# Token to use for the Identity service API.
CONFIG_KEYSTONE_ADMIN_TOKEN=c926f15bc0e54cc4b273e7467bd191de

# Email address for the Identity service 'admin' user. Defaults to:
CONFIG_KEYSTONE_ADMIN_EMAIL=root@localhost

# User name for the Identity service 'admin' user. Defaults to:
# 'admin'.
CONFIG_KEYSTONE_ADMIN_USERNAME=admin

# Password to use for the Identity service 'admin' user.
CONFIG_KEYSTONE_ADMIN_PW=lab

# Password to use for the Identity service 'demo' user.
CONFIG_KEYSTONE_DEMO_PW=lab
```

300 Appendix A: Sample Answer File for Packstack

```
# Identity service API version string (v2.0, v3).
CONFIG_KEYSTONE_API_VERSION=v2.0

# Identity service token format (UUID or PKI). The recommended format
# for new deployments is UUID.
CONFIG_KEYSTONE_TOKEN_FORMAT=UUID

# Name of service to use to run the Identity service (keystone,
# httpd).
CONFIG_KEYSTONE_SERVICE_NAME=httpd

# Type of Identity service backend (sql, ldap).
CONFIG_KEYSTONE_IDENTITY_BACKEND=sql

# URL for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_URL=ldap://192.168.1.22

# User DN for the Identity service LDAP backend. Used to bind to the
# LDAP server if the LDAP server does not allow anonymous
# authentication.
CONFIG_KEYSTONE_LDAP_USER_DN=

# User DN password for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_USER_PASSWORD=

# Base suffix for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_SUFFIX=

# Query scope for the Identity service LDAP backend. Use 'one' for
# onelevel/singleLevel or 'sub' for subtree/wholeSubtree ('base' is
# not actually used by the Identity service and is therefore
# deprecated) (base, one, sub)
CONFIG_KEYSTONE_LDAP_QUERY_SCOPE=one

# Query page size for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_PAGE_SIZE=-1

# User subtree for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_USER_SUBTREE=

# User query filter for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_USER_FILTER=

# User object class for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_USER_OBJECTCLASS=
```



```
# User ID attribute for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_USER_ID_ATTRIBUTE=

# User name attribute for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_USER_NAME_ATTRIBUTE=

# User email address attribute for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_USER_MAIL_ATTRIBUTE=

# User-enabled attribute for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_USER_ENABLED_ATTRIBUTE=

# Bit mask integer applied to user-enabled attribute for the Identity
# service LDAP backend. Indicate the bit that the enabled value is
# stored in if the LDAP server represents "enabled" as a bit on an
# integer rather than a boolean. A value of "0" indicates the mask is
# not used (default). If this is not set to "0", the typical value is
# "2", typically used when
# "CONFIG_KEYSTONE_LDAP_USER_ENABLED_ATTRIBUTE = userAccountControl".
CONFIG_KEYSTONE_LDAP_USER_ENABLED_MASK=-1

# Value of enabled attribute which indicates user is enabled for the
# Identity service LDAP backend. This should match an appropriate
# integer value if the LDAP server uses non-boolean (bitmask) values
# to indicate whether a user is enabled or disabled. If this is not
# set as 'y', the typical value is "512". This is typically used when
# "CONFIG_KEYSTONE_LDAP_USER_ENABLED_ATTRIBUTE = userAccountControl".
CONFIG_KEYSTONE_LDAP_USER_ENABLED_DEFAULT=TRUE

# Specify 'y' if users are disabled (not enabled) in the Identity
# service LDAP backend (inverts boolean-enabled values). Some LDAP
# servers use a boolean lock attribute where "y" means an account is
# disabled. Setting this to 'y' allows these lock attributes to be
# used. This setting will have no effect if
# "CONFIG_KEYSTONE_LDAP_USER_ENABLED_MASK" is in use (n, y).
CONFIG_KEYSTONE_LDAP_USER_ENABLED_INVERT=n

# Comma-separated list of attributes stripped from LDAP user entry
# upon update.
CONFIG_KEYSTONE_LDAP_USER_ATTRIBUTE_IGNORE=

# Identity service LDAP attribute mapped to default_project_id for
# users.
CONFIG_KEYSTONE_LDAP_USER_DEFAULT_PROJECT_ID_ATTRIBUTE=
```

```
# Specify 'y' if you want to be able to create Identity service users
# through the Identity service interface; specify 'n' if you will
# create directly in the LDAP backend (n, y).
CONFIG_KEYSTONE_LDAP_USER_ALLOW_CREATE=n

# Specify 'y' if you want to be able to update Identity service users
# through the Identity service interface; specify 'n' if you will
# update directly in the LDAP backend (n, y).
CONFIG_KEYSTONE_LDAP_USER_ALLOW_UPDATE=n

# Specify 'y' if you want to be able to delete Identity service users
# through the Identity service interface; specify 'n' if you will
# delete directly in the LDAP backend (n, y).
CONFIG_KEYSTONE_LDAP_USER_ALLOW_DELETE=n

# Identity service LDAP attribute mapped to password.
CONFIG_KEYSTONE_LDAP_USER_PASS_ATTRIBUTE=

# DN of the group entry to hold enabled LDAP users when using enabled
# emulation.
CONFIG_KEYSTONE_LDAP_USER_ENABLED_EMULATION_DN=

# List of additional LDAP attributes for mapping additional attribute
# mappings for users. The attribute-mapping format is
# <ldap_attr>:<user_attr>, where ldap_attr is the attribute in the
# LDAP entry and user_attr is the Identity API attribute.
CONFIG_KEYSTONE_LDAP_USER_ADDITIONAL_ATTRIBUTE_MAPPING=

# Group subtree for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_GROUP_SUBTREE=

# Group query filter for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_GROUP_FILTER=

# Group object class for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_GROUP_OBJECTCLASS=

# Group ID attribute for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_GROUP_ID_ATTRIBUTE=

# Group name attribute for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_GROUP_NAME_ATTRIBUTE=

# Group member attribute for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_GROUP_MEMBER_ATTRIBUTE=
```

```
# Group description attribute for the Identity service LDAP backend.
CONFIG_KEYSTONE_LDAP_GROUP_DESC_ATTRIBUTE=

# Comma-separated list of attributes stripped from LDAP group entry
# upon update.
CONFIG_KEYSTONE_LDAP_GROUP_ATTRIBUTE_IGNORE=

# Specify 'y' if you want to be able to create Identity service
# groups through the Identity service interface; specify 'n' if you
# will create directly in the LDAP backend (n, y).
CONFIG_KEYSTONE_LDAP_GROUP_ALLOW_CREATE=y

# Specify 'y' if you want to be able to update Identity service
# groups through the Identity service interface; specify 'n' if you
# will update directly in the LDAP backend (n, y).
CONFIG_KEYSTONE_LDAP_GROUP_ALLOW_UPDATE=y

# Specify 'y' if you want to be able to delete Identity service
# groups through the Identity service interface; specify 'n' if you
# will delete directly in the LDAP backend (n, y).
CONFIG_KEYSTONE_LDAP_GROUP_ALLOW_DELETE=y

# List of additional LDAP attributes used for mapping additional
# attribute mappings for groups. The attribute=mapping format is
# <ldap_attr>:<group_attr>, where ldap_attr is the attribute in the
# LDAP entry and group_attr is the Identity API attribute.
CONFIG_KEYSTONE_LDAP_GROUP_ADDITIONAL_ATTRIBUTE_MAPPING=

# Specify 'y' if the Identity service LDAP backend should use TLS (n,
# y).
CONFIG_KEYSTONE_LDAP_USE_TLS=n

# CA certificate directory for Identity service LDAP backend (if TLS
# is used).
CONFIG_KEYSTONE_LDAP_TLS_CACERTDIR=

# CA certificate file for Identity service LDAP backend (if TLS is
# used).
CONFIG_KEYSTONE_LDAP_TLS_CACERTFILE=

# Certificate-checking strictness level for Identity service LDAP
# backend (never, allow, demand).
CONFIG_KEYSTONE_LDAP_TLS_REQ_CERT=demand

# Password to use for the Image service (glance) to access the
```

```
# database.
CONFIG_GLANCE_DB_PW=e73ab8bc00a94083

# Password to use for the Image service to authenticate with the
# Identity service.
CONFIG_GLANCE_KS_PW=650e01a4fb1f457e

# Storage backend for the Image service (controls how the Image
# service stores disk images). Valid options are: file or swift
# (Object Storage). The Object Storage service must be enabled to use
# it as a working backend; otherwise, Packstack falls back to 'file'.
# ['file', 'swift']
CONFIG_GLANCE_BACKEND=file

# Password to use for the Block Storage service (cinder) to access
# the database.
CONFIG_CINDER_DB_PW=c0f4e8de016b4d76

# Password to use for the Block Storage service to authenticate with
# the Identity service.
CONFIG_CINDER_KS_PW=5e8fb5e4e7bd4b8f

# Storage backend to use for the Block Storage service; valid options
# are: lvm, gluster, nfs, vmdk, netapp. ['lvm', 'gluster', 'nfs',
# 'vmdk', 'netapp']
CONFIG_CINDER_BACKEND=lvm

# Specify 'y' to create the Block Storage volumes group. That is,
# Packstack creates a raw disk image in /var/lib/cinder, and mounts it
# using a loopback device. This should only be used for testing on a
# proof-of-concept installation of the Block Storage service (a file-
# backed volume group is not suitable for production usage) (y, n).
CONFIG_CINDER_VOLUMES_CREATE=y

# Size of Block Storage volumes group. Actual volume size will be
# extended with 3% more space for VG metadata. Remember that the size
# of the volume group will restrict the amount of disk space that you
# can expose to Compute instances, and that the specified amount must
# be available on the device used for /var/lib/cinder.
CONFIG_CINDER_VOLUMES_SIZE=1400G

# A single or comma-separated list of Red Hat Storage (gluster)
# volume shares to mount. Example: 'ip-address:/vol-name', 'domain
# :/vol-name'
CONFIG_CINDER_GLUSTER_MOUNTS=
```

```
# A single or comma-separated list of NFS exports to mount. Example:
# 'ip-address:/export-name'
CONFIG_CINDER_NFS_MOUNTS=

# Administrative user account name used to access the NetApp storage
# system or proxy server.
CONFIG_CINDER_NETAPP_LOGIN=

# Password for the NetApp administrative user account specified in
# the CONFIG_CINDER_NETAPP_LOGIN parameter.
CONFIG_CINDER_NETAPP_PASSWORD=

# Hostname (or IP address) for the NetApp storage system or proxy
# server.
CONFIG_CINDER_NETAPP_HOSTNAME=

# The TCP port to use for communication with the storage system or
# proxy. If not specified, Data ONTAP drivers will use 80 for HTTP and
# 443 for HTTPS; E-Series will use 8080 for HTTP and 8443 for HTTPS.
# Defaults to: 80.
CONFIG_CINDER_NETAPP_SERVER_PORT=80

# Storage family type used on the NetApp storage system; valid
# options are ontap_7mode for using Data ONTAP operating in 7-Mode,
# ontap_cluster for using clustered Data ONTAP, or E-Series for NetApp
# E-Series. Defaults to: ontap_cluster. ['ontap_7mode',
# 'ontap_cluster', 'eseries']
CONFIG_CINDER_NETAPP_STORAGE_FAMILY=ontap_cluster

# The transport protocol used when communicating with the NetApp
# storage system or proxy server. Valid values are http or https.
# Defaults to: 'http' ('http', 'https').
CONFIG_CINDER_NETAPP_TRANSPORT_TYPE=http

# Storage protocol to be used on the data path with the NetApp
# storage system; valid options are iscsi, fc, nfs. Defaults to: nfs
# (iscsi, fc, nfs).
CONFIG_CINDER_NETAPP_STORAGE_PROTOCOL=nfs

# Quantity to be multiplied by the requested volume size to ensure
# enough space is available on the virtual storage server (Vserver) to
# fulfill the volume creation request. Defaults to: 1.0.
CONFIG_CINDER_NETAPP_SIZE_MULTIPLIER=1.0

# Time period (in minutes) that is allowed to elapse after the image
```

```
# is last accessed, before it is deleted from the NFS image cache.
# When a cache-cleaning cycle begins, images in the cache that have
# not been accessed in the last M minutes, where M is the value of
# this parameter, are deleted from the cache to create free space on
# the NFS share. Defaults to: 720.
CONFIG_CINDER_NETAPP_EXPIRY_THRES_MINUTES=720

# If the percentage of available space for an NFS share has dropped
# below the value specified by this parameter, the NFS image cache is
# cleaned. Defaults to: 20.
CONFIG_CINDER_NETAPP_THRES_AVL_SIZE_PERC_START=20

# When the percentage of available space on an NFS share has reached
# the percentage specified by this parameter, the driver stops
# clearing files from the NFS image cache that have not been accessed
# in the last M minutes, where M is the value of the
# CONFIG_CINDER_NETAPP_EXPIRY_THRES_MINUTES parameter. Defaults to:
# 60.
CONFIG_CINDER_NETAPP_THRES_AVL_SIZE_PERC_STOP=60

# Single or comma-separated list of NetApp NFS shares for Block
# Storage to use. Format: ip-address:/export-name. Defaults to: ''.
CONFIG_CINDER_NETAPP_NFS_SHARES=

# File with the list of available NFS shares. Defaults to:
# '/etc/cinder/shares.conf'.
CONFIG_CINDER_NETAPP_NFS_SHARES_CONFIG=/etc/cinder/shares.conf

# This parameter is only utilized when the storage protocol is
# configured to use iSCSI or FC. This parameter is used to restrict
# provisioning to the specified controller volumes. Specify the value
# of this parameter to be a comma separated list of NetApp controller
# volume names to be used for provisioning. Defaults to: ''.
CONFIG_CINDER_NETAPP_VOLUME_LIST=

# The vFiler unit on which provisioning of block storage volumes will
# be done. This parameter is only used by the driver when connecting
# to an instance with a storage family of Data ONTAP operating in
# 7-Mode Only use this parameter when utilizing the MultiStore feature
# on the NetApp storage system. Defaults to: ''.
CONFIG_CINDER_NETAPP_VFILER=

# The name of the config.conf stanza for a Data ONTAP (7-mode) HA
# partner. This option is only used by the driver when connecting to
# an instance with a storage family of Data ONTAP operating in 7-Mode,
```

```
# and it is required if the storage protocol selected is FC. Defaults
# to: ''.
CONFIG_CINDER_NETAPP_PARTNER_BACKEND_NAME=

# This option specifies the virtual storage server (Vserver) name on
# the storage cluster on which provisioning of block storage volumes
# should occur. Defaults to: ''.
CONFIG_CINDER_NETAPP_VSERVER=

# Restricts provisioning to the specified controllers. Value must be
# a comma-separated list of controller hostnames or IP addresses to be
# used for provisioning. This option is only utilized when the storage
# family is configured to use E-Series. Defaults to: ''.
CONFIG_CINDER_NETAPP_CONTROLLER_IPS=

# Password for the NetApp E-Series storage array. Defaults to: ''.
CONFIG_CINDER_NETAPP_SA_PASSWORD=

# This option is used to define how the controllers in the E-Series
# storage array will work with the particular operating system on the
# hosts that are connected to it. Defaults to: 'linux_dm_mp'
CONFIG_CINDER_NETAPP_ESERIES_HOST_TYPE=linux_dm_mp

# Path to the NetApp E-Series proxy application on a proxy server.
# The value is combined with the value of the
# CONFIG_CINDER_NETAPP_TRANSPORT_TYPE, CONFIG_CINDER_NETAPP_HOSTNAME,
# and CONFIG_CINDER_NETAPP_HOSTNAME options to create the URL used by
# the driver to connect to the proxy application. Defaults to:
# '/devmgr/v2'.
CONFIG_CINDER_NETAPP_WEBSERVICE_PATH=/devmgr/v2

# Restricts provisioning to the specified storage pools. Only dynamic
# disk pools are currently supported. The value must be a comma-
# separated list of disk pool names to be used for provisioning.
# Defaults to: ''.
CONFIG_CINDER_NETAPP_STORAGE_POOLS=

# Password to use for the OpenStack File Share service (manila) to
# access the database.
CONFIG_MANILA_DB_PW=lab

# Password to use for the OpenStack File Share service (manila) to
# authenticate with the Identity service.
CONFIG_MANILA_KS_PW=lab
```

```

# Backend for the OpenStack File Share service (manila); valid
# options are: generic or netapp (generic, netapp).
CONFIG_MANILA_BACKEND=generic

# Denotes whether the driver should handle the responsibility of
# managing share servers. This must be set to false if the driver is
# to operate without managing share servers. Defaults to: 'false'
# (true, false).
CONFIG_MANILA_NETAPP_DRV_HANDLES_SHARE_SERVERS=false

# The transport protocol used when communicating with the storage
# system or proxy server. Valid values are 'http' and 'https'.
# Defaults to: 'https' (https, http).
CONFIG_MANILA_NETAPP_TRANSPORT_TYPE=https

# Administrative user account name used to access the NetApp storage
# system. Defaults to: ''.
CONFIG_MANILA_NETAPP_LOGIN=admin

# Password for the NetApp administrative user account specified in
# the CONFIG_MANILA_NETAPP_LOGIN parameter. Defaults to: ''.
CONFIG_MANILA_NETAPP_PASSWORD=

# Hostname (or IP address) for the NetApp storage system or proxy
# server. Defaults to: ''.
CONFIG_MANILA_NETAPP_SERVER_HOSTNAME=

# The storage family type used on the storage system; valid values
# are ontap_cluster for clustered Data ONTAP. Defaults to:
# 'ontap_cluster'.
CONFIG_MANILA_NETAPP_STORAGE_FAMILY=ontap_cluster

# The TCP port to use for communication with the storage system or
# proxy server. If not specified, Data ONTAP drivers will use 80 for
# HTTP and 443 for HTTPS. Defaults to: '443'.
CONFIG_MANILA_NETAPP_SERVER_PORT=443

# Pattern for searching available aggregates for NetApp provisioning.
# Defaults to: '(.*)'.
CONFIG_MANILA_NETAPP_AGGREGATE_NAME_SEARCH_PATTERN=(.*)

# Name of aggregate on which to create the NetApp root volume. This
# option only applies when the option
# CONFIG_MANILA_NETAPP_DRV_HANDLES_SHARE_SERVERS is set to True.
CONFIG_MANILA_NETAPP_ROOT_VOLUME_AGGREGATE=

```



```
# NetApp root volume name. Defaults to: 'root'.
CONFIG_MANILA_NETAPP_ROOT_VOLUME_NAME=root

# This option specifies the storage virtual machine (previously
# called a Vserver) name on the storage cluster on which provisioning
# of shared file systems should occur. This option only applies when
# the option driver_handles_share_servers is set to False. Defaults
# to: ''.
CONFIG_MANILA_NETAPP_VSERVER=

# Denotes whether the driver should handle the responsibility of
# managing share servers. This must be set to false if the driver is
# to operate without managing share servers. Defaults to: 'true'.
# ['true', 'false']
CONFIG_MANILA_GENERIC_DRV_HANDLES_SHARE_SERVERS=true

# Volume name template for Manila service. Defaults to: 'manila-
# share-%s'.
CONFIG_MANILA_GENERIC_VOLUME_NAME_TEMPLATE=manila-share-%s

# Share mount path for Manila service. Defaults to: '/shares'.
CONFIG_MANILA_GENERIC_SHARE_MOUNT_PATH=/shares

# Location of disk image for Manila service instance. Defaults to: '
CONFIG_MANILA_SERVICE_IMAGE_LOCATION=https://www.dropbox.com/s/vi5oeh10q1qkckh/
ubuntu_1204_nfs_cifs.qcow2

# User in Manila service instance.
CONFIG_MANILA_SERVICE_INSTANCE_USER=ubuntu

# Password to service instance user.
CONFIG_MANILA_SERVICE_INSTANCE_PASSWORD=ubuntu

# Type of networking that the backend will use. A more detailed
# description of each option is available in the Manila docs. Defaults
# to: 'neutron'. ['neutron', 'nova-network', 'standalone']
CONFIG_MANILA_NETWORK_TYPE=neutron

# Gateway IPv4 address that should be used. Required. Defaults to:
# ''.
CONFIG_MANILA_NETWORK_STANDALONE_GATEWAY=

# Network mask that will be used. Can be either decimal like '24' or
# binary like '255.255.255.0'. Required. Defaults to: ''.
CONFIG_MANILA_NETWORK_STANDALONE_NETMASK=
```

310 Appendix A: Sample Answer File for Packstack

```
# Set it if network has segmentation (VLAN, VXLAN, etc). It will be
# assigned to share-network and share drivers will be able to use this
# for network interfaces within provisioned share servers. Optional.
# Example: 1001. Defaults to: ''.
CONFIG_MANILA_NETWORK_STANDALONE_SEG_ID=

# Can be IP address, range of IP addresses or list of addresses or
# ranges. Contains addresses from IP network that are allowed to be
# used. If empty, then will be assumed that all host addresses from
# network can be used. Optional. Examples: 10.0.0.10 or
# 10.0.0.10-10.0.0.20 or
# 10.0.0.10-10.0.0.20,10.0.0.30-10.0.0.40,10.0.0.50. Defaults to: ''.
CONFIG_MANILA_NETWORK_STANDALONE_IP_RANGE=

# IP version of network. Optional. Defaults to: 4 (4, 6).
CONFIG_MANILA_NETWORK_STANDALONE_IP_VERSION=4

# Password to use for OpenStack Bare Metal Provisioning (ironic) to
# access the database.
CONFIG_IRONIC_DB_PW=lab

# Password to use for OpenStack Bare Metal Provisioning to
# authenticate with the Identity service.
CONFIG_IRONIC_KS_PW=lab

# Password to use for the Compute service (nova) to access the
# database.
CONFIG_NOVA_DB_PW=a3f51ff28de54548

# Password to use for the Compute service to authenticate with the
# Identity service.
CONFIG_NOVA_KS_PW=0a2872f1b1d94818

# Overcommitment ratio for virtual to physical CPUs. Specify 1.0 to
# disable CPU overcommitment.
CONFIG_NOVA_SCHED_CPU_ALLOC_RATIO=16.0

# Overcommitment ratio for virtual to physical RAM. Specify 1.0 to
# disable RAM overcommitment.
CONFIG_NOVA_SCHED_RAM_ALLOC_RATIO=1.5

# Protocol used for instance migration. Valid options are: tcp and
# ssh. Note that by default, the Compute user is created with the
# /sbin/nologin shell so that the SSH protocol will not work. To make
# the SSH protocol work, you must configure the Compute user on
```

```
# compute hosts manually (tcp, ssh).
CONFIG_NOVA_COMPUTE_MIGRATE_PROTOCOL=tcp

# Manager that runs the Compute service.
CONFIG_NOVA_COMPUTE_MANAGER=nova.compute.manager.ComputeManager

# PEM encoded certificate to be used for ssl on the https server,
# leave blank if one should be generated, this certificate should not
# require a passphrase. If CONFIG_HORIZON_SSL is set to 'n' this
# parameter is ignored.
CONFIG_VNC_SSL_CERT=

# SSL keyfile corresponding to the certificate if one was entered. If
# CONFIG_HORIZON_SSL is set to 'n' this parameter is ignored.
CONFIG_VNC_SSL_KEY=

# Private interface for flat DHCP on the Compute servers.
CONFIG_NOVA_COMPUTE_PRIVIF=eth1.10

# Compute Network Manager. ['^nova\.network\.manager\.w+Manager$']
CONFIG_NOVA_NETWORK_MANAGER=nova.network.manager.FlatDHCPManager

# Public interface on the Compute network server.
CONFIG_NOVA_NETWORK_PUBIF=eth0

# Private interface for flat DHCP on the Compute network server.
CONFIG_NOVA_NETWORK_PRIVIF=eth1.10

# IP Range for flat DHCP. ['^[\:\.\da-fA-f]+(\\/\d+){0,1}$']
CONFIG_NOVA_NETWORK_FIXEDRANGE=192.168.1.0/24

# IP Range for floating IP addresses. ['^[\:\.\da-
# fA-f]+(\\/\d+){0,1}$']
CONFIG_NOVA_NETWORK_FLOATRANGE=192.168.2.0/24

# Specify 'y' to automatically assign a floating IP to new instances.
# (y, n)
CONFIG_NOVA_NETWORK_AUTOASSIGNFLOATINGIP=n

# First VLAN for private networks (Compute networking).
CONFIG_NOVA_NETWORK_VLAN_START=100

# Number of networks to support (Compute networking).
CONFIG_NOVA_NETWORK_NUMBER=1
```

312 Appendix A: Sample Answer File for Packstack

```
# Number of addresses in each private subnet (Compute networking).
CONFIG_NOVA_NETWORK_SIZE=255

# Password to use for OpenStack Networking (neutron) to authenticate
# with the Identity service.
CONFIG_NEUTRON_KS_PW=2820c6f0c1314209

# The password to use for OpenStack Networking to access the
# database.
CONFIG_NEUTRON_DB_PW=fd1662e09a6f4165

# The name of the Open vSwitch bridge (or empty for linuxbridge) for
# the OpenStack Networking L3 agent to use for external traffic.
# Specify 'provider' if you intend to use a provider network to handle
# external traffic.
CONFIG_NEUTRON_L3_EXT_BRIDGE=br-ex

# Password for the OpenStack Networking metadata agent.
CONFIG_NEUTRON_METADATA_PW=3658c5fbcad74f6e

# Specify 'y' to install OpenStack Networking's Load-Balancing-
# as-a-Service (LBaaS) (y, n).
CONFIG_LBAAS_INSTALL=y

# Specify 'y' to install OpenStack Networking's L3 Metering agent (y,
# n).
CONFIG_NEUTRON_METERING_AGENT_INSTALL=y

# Specify 'y' to configure OpenStack Networking's Firewall-
# as-a-Service (FWaaS) (y, n)
CONFIG_NEUTRON_FWAAS=y

# Comma-separated list of network-type driver entry points to be
# loaded from the neutron.ml2.type_drivers namespace (local, flat,
# vlan, gre, vxlan).
CONFIG_NEUTRON_ML2_TYPE_DRIVERS=local,vlan,flat,gre,vxlan

# Comma-separated, ordered list of network types to allocate as
# tenant networks. The 'local' value is only useful for single-box
# testing and provides no connectivity between hosts (local, vlan,
# gre, vxlan).
CONFIG_NEUTRON_ML2_TENANT_NETWORK_TYPES=vlan

# Comma-separated ordered list of networking mechanism driver entry
# points to be loaded from the neutron.ml2.mechanism_drivers namespace
```

```

# (logger, test, linuxbridge, openvswitch, hyperv, ncs, arista,
# cisco_nexus, mlnx, l2population).
CONFIG_NEUTRON_ML2_MECHANISM_DRIVERS=openvswitch

# Comma-separated list of physical_network names with which flat
# networks can be created. Use * to allow flat networks with arbitrary
# physical_network names.
CONFIG_NEUTRON_ML2_FLAT_NETWORKS=*

# Comma-separated list of <physical_network>:<vlan_min>:<vlan_max> or
# <physical_network> specifying physical_network names usable for VLAN
# provider and tenant networks, as well as ranges of VLAN tags on each
# available for allocation to tenant networks.
CONFIG_NEUTRON_ML2_VLAN_RANGES=physnet1:2:4094

# Comma-separated list of <tun_min>:<tun_max> tuples enumerating
# ranges of GRE tunnel IDs that are available for tenant-network
# allocation. A tuple must be an array with tun_max +1 - tun_min >
# 1000000.
CONFIG_NEUTRON_ML2_TUNNEL_ID_RANGES=

# Comma-separated list of addresses for VXLAN multicast group. If
# left empty, disables VXLAN from sending allocate broadcast traffic
# (disables multicast VXLAN mode). Should be a Multicast IP (v4 or v6)
# address.
CONFIG_NEUTRON_ML2_VXLAN_GROUP=

# Comma-separated list of <vni_min>:<vni_max> tuples enumerating
# ranges of VXLAN VNI IDs that are available for tenant network
# allocation. Minimum value is 0 and maximum value is 16777215.
CONFIG_NEUTRON_ML2_VNI_RANGES=5001:10000

# Name of the L2 agent to be used with OpenStack Networking
# (linuxbridge, openvswitch).
CONFIG_NEUTRON_L2_AGENT=openvswitch

# Comma-separated list of interface mappings for the OpenStack
# Networking linuxbridge plugin. Each tuple in the list must be in the
# format <physical_network>:<net_interface>. Example:
# physnet1:eth1,physnet2:eth2,physnet3:eth3.
CONFIG_NEUTRON_LB_INTERFACE_MAPPINGS=

# Comma-separated list of bridge mappings for the OpenStack
# Networking Open vSwitch plugin. Each tuple in the list must be in
# the format <physical_network>:<ovs_bridge>. Example: physnet1:br-

```

```

# eth1,physnet2:br-eth2,physnet3:br-eth3
CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS=physnet1:br-data

# Comma-separated list of colon-separated Open vSwitch
# <bridge>:<interface> pairs. The interface will be added to the
# associated bridge. If you desire the bridge to be persistent a value
# must be added to this directive, also
# CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS must be set in order to create
# the proper port. This can be achieved from the command line by
# issuing the following command: packstack --allinone --os-neutron-
# ovs-bridge-mappings=ext-net:br-ex --os-neutron-ovs-bridge-interfaces
# =br-ex:eth0
CONFIG_NEUTRON_OVS_BRIDGE_IFACES=br-data:eth1.10

# Interface for the Open vSwitch tunnel. Packstack overrides the IP
# address used for tunnels on this hypervisor to the IP found on the
# specified interface (for example, eth1).
CONFIG_NEUTRON_OVS_TUNNEL_IF=

# VXLAN UDP port.
CONFIG_NEUTRON_OVS_VXLAN_UDP_PORT=4789

# Specify 'y' to set up Horizon communication over https (y, n).
CONFIG_HORIZON_SSL=n

# Secret key to use for Horizon Secret Encryption Key.
CONFIG_HORIZON_SECRET_KEY=93029ad02c8f49f8865f0e6a41b8f8f4

# PEM-encoded certificate to be used for SSL connections on the https
# server (the certificate should not require a passphrase). To
# generate a certificate, leave blank.
CONFIG_HORIZON_SSL_CERT=

# SSL keyfile corresponding to the certificate if one was specified.
CONFIG_HORIZON_SSL_KEY=

CONFIG_HORIZON_SSL_CACERT=

# Password to use for the Object Storage service to authenticate with
# the Identity service.
CONFIG_SWIFT_KS_PW=8b3a14537ffa4c37

# Comma-separated list of devices to use as storage device for Object
# Storage. Each entry must take the format /path/to/dev (for example,
# specifying /dev/vdb installs /dev/vdb as the Object Storage storage

```

```
# device; Packstack does not create the filesystem, you must do this
# first). If left empty, Packstack creates a loopback device for test
# setup.
CONFIG_SWIFT_STORAGES=

# Number of Object Storage storage zones; this number MUST be no
# larger than the number of configured storage devices.
CONFIG_SWIFT_STORAGE_ZONES=1

# Number of Object Storage storage replicas; this number MUST be no
# larger than the number of configured storage zones.
CONFIG_SWIFT_STORAGE_REPLICAS=1

# File system type for storage nodes (xfs, ext4).
CONFIG_SWIFT_STORAGE_FSTYPE=ext4

# Custom seed number to use for swift_hash_path_suffix in
# /etc/swift/swift.conf. If you do not provide a value, a seed number
# is automatically generated.
CONFIG_SWIFT_HASH=8af0d28623254c73

# Size of the Object Storage loopback file storage device.
CONFIG_SWIFT_STORAGE_SIZE=2G

# Password used by Orchestration service user to authenticate against
# the database.
CONFIG_HEAT_DB_PW=lab

# Encryption key to use for authentication in the Orchestration
# database (16, 24, or 32 chars).
CONFIG_HEAT_AUTH_ENC_KEY=bef72d0287344cc8

# Password to use for the Orchestration service to authenticate with
# the Identity service.
CONFIG_HEAT_KS_PW=lab

# Specify 'y' to install the Orchestration CloudWatch API (y, n).
CONFIG_HEAT_CLOUDWATCH_INSTALL=n

# Specify 'y' to install the Orchestration CloudFormation API (y, n).
CONFIG_HEAT_CFN_INSTALL=n

# Name of the Identity domain for Orchestration.
CONFIG_HEAT_DOMAIN=heat
```

316 Appendix A: Sample Answer File for Packstack

```
# Name of the Identity domain administrative user for Orchestration.
CONFIG_HEAT_DOMAIN_ADMIN=heat_admin

# Password for the Identity domain administrative user for
# Orchestration.
CONFIG_HEAT_DOMAIN_PASSWORD=lab

# Specify 'y' to provision for demo usage and testing (y, n).
CONFIG_PROVISION_DEMO=y

# Specify 'y' to configure the OpenStack Integration Test Suite
# (tempest) for testing. The test suite requires OpenStack Networking
# to be installed (y, n).
CONFIG_PROVISION_TEMPEST=y

# CIDR network address for the floating IP subnet.
CONFIG_PROVISION_DEMO_FLOATRANGE=172.24.4.224/28

# The name to be assigned to the demo image in Glance (default
# "cirros").
CONFIG_PROVISION_IMAGE_NAME=cirros

# A URL or local file location for an image to download and provision
# in Glance (defaults to a URL for a recent "cirros" image).
CONFIG_PROVISION_IMAGE_URL=http://download.cirros-cloud.net/0.3.3/cirros-0.3.3-
x86_64-disk.img

# Format for the demo image (default "qcow2").
CONFIG_PROVISION_IMAGE_FORMAT=qcow2

# User to use when connecting to instances booted from the demo
# image.
CONFIG_PROVISION_IMAGE_SSH_USER=cirros

# Name of the Integration Test Suite provisioning user. If you do not
# provide a user name, Tempest is configured in a standalone mode.
CONFIG_PROVISION_TEMPEST_USER=

# Password to use for the Integration Test Suite provisioning user.
CONFIG_PROVISION_TEMPEST_USER_PW=lab

# CIDR network address for the floating IP subnet.
CONFIG_PROVISION_TEMPEST_FLOATRANGE=172.24.4.224/28

# URI of the Integration Test Suite git repository.
```



```
CONFIG_PROVISION_TEMPEST_REPO_URI=https://github.com/openstack/tempest.git

# Revision (branch) of the Integration Test Suite git repository.
CONFIG_PROVISION_TEMPEST_REPO_REVISION=master

# Specify 'y' to configure the Open vSwitch external bridge for an
# all-in-one deployment (the L3 external bridge acts as the gateway
# for virtual machines) (y, n).
CONFIG_PROVISION_ALL_IN_ONE_OVS_BRIDGE=n

# Secret key for signing Telemetry service (ceilometer) messages.
CONFIG_CEILOMETER_SECRET=b86ce657da7e4eaa

# Password to use for Telemetry to authenticate with the Identity
# service.
CONFIG_CEILOMETER_KS_PW=d563856ba5fd47c6

# Backend driver for Telemetry's group membership coordination
# (redis, none).
CONFIG_CEILOMETER_COORDINATION_BACKEND=redis

# IP address of the server on which to install MongoDB.
CONFIG_MONGODB_HOST=192.168.1.22

# IP address of the server on which to install the Redis master
# server.
CONFIG_REDIS_MASTER_HOST=192.168.1.22

# Port on which the Redis server(s) listens.
CONFIG_REDIS_PORT=6379

# Specify 'y' to have Redis try to use HA (y, n).
CONFIG_REDIS_HA=n

# Hosts on which to install Redis slaves.
CONFIG_REDIS_SLAVE_HOSTS=

# Hosts on which to install Redis sentinel servers.
CONFIG_REDIS_SENTINEL_HOSTS=

# Host to configure as the Redis coordination sentinel.
CONFIG_REDIS_SENTINEL_CONTACT_HOST=

# Port on which Redis sentinel servers listen.
CONFIG_REDIS_SENTINEL_PORT=26379
```

```
# Quorum value for Redis sentinel servers.
CONFIG_REDIS_SENTINEL_QUORUM=2

# Name of the master server watched by the Redis sentinel (eg.
# master).
CONFIG_REDIS_MASTER_NAME=mymaster

# Password to use for OpenStack Data Processing (sahara) to access
# the database.
CONFIG_SAHARA_DB_PW=lab

# Password to use for OpenStack Data Processing to authenticate with
# the Identity service.
CONFIG_SAHARA_KS_PW=lab

# Password to use for OpenStack Database-as-a-Service (trove) to
# access the database.
CONFIG_TROVE_DB_PW=lab

# Password to use for OpenStack Database-as-a-Service to authenticate
# with the Identity service.
CONFIG_TROVE_KS_PW=lab

# User name to use when OpenStack Database-as-a-Service connects to
# the Compute service.
CONFIG_TROVE_NOVA_USER=admin

# Tenant to use when OpenStack Database-as-a-Service connects to the
# Compute service.
CONFIG_TROVE_NOVA_TENANT=services

# Password to use when OpenStack Database-as-a-Service connects to
# the Compute service.
CONFIG_TROVE_NOVA_PW=lab

# Password of the nagiosadmin user on the Nagios server.
CONFIG_NAGIOS_PW=16ff1ac86832473b
```

Index

A

A9 processor, 1

active/active flow distribution to
 cloudburst, redirection access
 model, 193

adoption, Enterprise cloud, 29-30

algorithms, scheduling, 60-62

Amazon web services deployment, CSR,
 211-215

Amazon web services deployment, CSR
 1000V, 216-222

application-centric infrastructure
 (SDN), 224

Application Virtual Switch (AVS), 270

architecture, CSR 1000V, troubleshooting,
 271-272

ASR (Aggregation Service Router) 1000
 router, 41, 96-97

 architectural elements, 95

 ESPs (embedded service processors),
 42-43

 RP (route processor), 42

 SIP (SPA interface processor), 43

ASR (Aggregation Service Router)
 1001, 97

 logical architecture, 97

 virtualizing into CSR 1000V, 98

ATM (Asynchronous Transfer Mode), 13

attach-device command (virsh), 90

attach-disk command (virsh), 90

attach-interface command (virsh), 90

automation, 247-248

 BDEO tool, 248-249

 management, 247

 NSO tool, 249-251

NFV orchestration with OpenStack,
 252-253, 260-261, 264-266

 versus orchestration, 247

 provisioning, 247

availability zones, VPC, 214

AVC (Application Visibility and Control),
 CSR 1000V, 52-53

AVS (Application Virtual Switch), 270

AWS (Amazon web services) deployment
 CSR, 211-215

 CSR 1000V, 216-222

B

BadUidbSubIdx drop type (IOS), 285

Basic Input/Output System (BIOS), 67

BDEO tool, 248-249

BIOS (Basic Input/Output System), 67

BIOS settings, hosts, 276

block-based storage, 3

boot process (IOS), 66-67

BPG route reflector, CSR, 155-157

 hierarchical use, 157-162

BqsOor drop type (IOS), 285

branch design, CSR 1000V, planning,
 162-168

C

- caching memory, Linux, 71
- Ceilometer project, OpenStack, 205
- chunk manager (memory manager), 66
- CIM (Common Information Model) system
 - processes, VMkernel, 77-78
- Cinder component (OpenStack), 34
- Cinder project, OpenStack, 204
- Cisco Domain 10 framework, 22
 - abstraction and virtualization, 23
 - automation and orchestration, 23
 - customer interface, 24
 - infrastructure and environmental, 22-23
 - organization, governance, and process, 25-26
 - platform and application, 24
 - security and compliance, 24-25
 - service catalog and financials, 24
- Cisco inter-cloud fabric, CSR 1000V
 - cloudburst, 194-195
- CLI (command-line interface)
 - control-processor output, 282
 - show interfaces command output, 282
 - sw-nic output, 288
- Client, CSR data plane, 101
- clientless mode (SSL VPNs), 147
- cloud computing, 1
 - design, 21-23
 - on-demand service*, 21
 - Enterprise
 - adoption challenges*, 29-30
 - connectivity*, 26-28
- cloud deployment models, 20-21
- cloud services
 - IaaS (Infrastructure as a Service), 18-19
 - PaaS (Platform as a Service), 19
 - SaaS (Software as a Service), 20
- cloudburst, CSR 1000V, 190
 - Cisco inter-cloud fabric, 194-195
 - data synchronization, 191
 - direct access model, 191-192
 - network connectivity, 190
 - redirection access model, 192-193
 - workload migration, 191
- code listings
 - Changing the Speed of the Interface, 279
 - Configuration Script Sample, 240
 - Control-Processor CLI Output, 282
 - CSR 1000V as a DMVPN Hub, 149-150
 - CSR as a Remote Access VPN Server with an AnyConnect Client, 153-155
 - ESXi NIC Stats, 290
 - ESXi Port Stats, 292
 - Example of Using the glance CLI to Add a CSR 1000V Image, 236
 - How to Check Throughput Levels and License Details, 277
 - Interface Controller Output, 283-285
 - Interface show Command, 280
 - ISR as a DMVPN Spoke, 150-152
 - Kernel Images Available to GRUB Are Listed in menu.lst, 67-68
 - LMGW Configuration, 177-179
 - MS/MR Configuration, 179-181
 - Programs That Are Executed on the Full-Multiuser Run Level, 69
 - QFP Feature Debugging Options, 286-288
 - R1 Configuration, 159
 - R2 Configuration, 159
 - R3 Configuration, 161
 - R4 Configuration, 161
 - R5-1 Configuration, 160
 - R5-2 Configuration, 160
 - RR1 Configuration, 158
 - RR2 Configuration, 158
 - RR-3 Configuration, 160
 - RR-4 Configuration, 161
 - Sample Answer File for Packstack, 231-233
 - Sample Definition of VNF Descriptors, 261-264
 - Sample NSO Initiation to Understand Input to Be Used in Service and Device Model Framework, 253-260
 - Sample of Installed NED Verification, 260
 - Sample VNF Instantiation, 264
 - show Commands, 182
 - show interfaces CLI Output, 283
 - Snapshot of BGP Update at R4, 162
 - sw-nic CLI Output, 288
 - VM List on an ESXi Host, 289
- commands
 - show, 181-182, 278
 - show interface, 279

- show interface controller, 283
- show interfaces, 282
- speed, 278
- statistics drop, 285
- computation, data centers, 3**
- conceptual architecture, data center virtualization, 5-6**
- configuration**
 - CPU usage, 281-282
 - hardware and software speed, 278-279
 - hosts, 275-276
 - interface-to-NIC mapping, 281
 - memory usage, 282-288
- connectivity, Enterprise cloud, 26-28**
- containers, 8**
- control planes, LISP, 171-175**
- controlled resources, 83**
- core partitioning, hypervisors, 75**
- CPUs**
 - pining, 138
 - settings, hosts, 275
 - scheduling algorithms, 60-62
 - usage configuration, 281-282
- CPU scheduler, VMkernel, 76**
- create command (virsh), 89**
- Create Router dialog (OpenStack), 238**
- critical priority, ready queue, 65**
- Critical queue (IOS scheduler), 38**
- crypto engine, CSR 1000V, 103**
- crypto maps, IPsec VPNs, 143-144**
- CSR (cloud service router)**
 - BPG route reflector, 155-157
 - hierarchical use, 157-162*
 - host configuration, 275-276
 - hypervisors, 59
 - LISP (Locator/ID Separation Protocol), 168-169, 175
 - control plane, 171-175*
 - data plane, 171*
 - ETR (egress tunnel router), 169-170*
 - IP mobility, 175*
 - IPv6 migration, 175*
 - ITR (ingress tunnel router), 169-170*
 - MR (MAP Resolver), 170*
 - network-to-network connectivity, 175-176*
 - network-to-network interconnection configuration, 176-182*
 - PETR (proxy egress tunnel router), 170-171*
 - PITR (proxy ingress tunnel router), 170*
- OpenStack
 - instantiating Neutron plugin, 242-245*
 - tenant deployment, 235-242*
- public cloud deployment, Amazon web services, 211-215
- as Remote Access VPN server, 153-155
- remote VPN access into Cloud, 153-155
- secure inter-cloud connectivity, 152
- CSR 1000V, 37, 95**
 - AVC (Application Visibility and Control), 52-53
 - branch design planning, 162-164
 - virtualization, 164-168*
 - cloudburst, 190
 - Cisco inter-cloud fabric, 194-195*
 - data synchronization, 191*
 - direct access model, 191-192*
 - network connectivity, 190*
 - redirection access model, 192-193*
 - workload migration, 191*
 - data plane, 103-104
 - architecture, 100-102*
 - Netmap I/O, 104-105*
 - packet flow, 106-109*
 - DMVPN (Dynamic Multipoint VPN), 53
 - EEM (Embedded Event Manager), 54
 - initiation process, 99-100
 - installing
 - KVM hypervisor, 126-137*
 - VMware hypervisor, 110-125*
 - IP SLA (IP Service Level Agreement), 54
 - LISP (Location/ID Separation Protocol), 54
 - MPLS (Multiprotocol Label Switching) VPN, 54-55
 - multitenant data center, 185-190
 - zone connectivity, 188*
 - as Neutron router, 206-209
 - OTV (Overlay Transport Virtualization), 55
 - performance tuning, 137-139
 - PfR (Performance Routing), 55
 - private cloud deployment in OpenStack, 195-211

- public cloud deployment, 211
 - Amazon web services, 216-222*
- Radio Aware Routing, 56
- Redundancy Group Infrastructure, 56
- software crypto engine, 103
- system design, 95-98
- as tenant router, 209-211
- troubleshooting, 271
 - architecture overview, 271-272*
 - debugging packet loss, 276-292*
 - I/O configuration, 272-276*
- virtualizing ASR 1001 into, 98
- VM (virtual machine), 271
 - layers, 103*
- VPLS (Virtual Private LAN Services), 55
- VPN service gateway, 148-153
- VPN services, 141
 - L2VPNs, 141*
 - L3VPNs, 142-148*
- VXLAN (Virtual Extensible LAN), 56
- ZBFW (Zone Based Firewall), 56-57
- CSR 1000V routers, 44-45**
 - deployment requirements, 45-47
 - elastic performance and scaling, 47-48
 - network extension from premises to cloud, 51
 - rapid deployment, 49
 - routing flexibility, 49
 - secure cloud VPN gateway, 50-51
 - segmentation within cloud, 52
- CSRVR, bringing up as guest, 126-137**

D

- daemon (IOSd), 40-41**
- DAL (Database Abstraction Layer), 202**
- data, unstructured, 4**
- data centers, 1-2**
 - computation, 3
 - distributed servers, 2
 - evolution, 2
 - facilities, 3
 - multitenant, 16-18
 - CSR 1000V, 185-190*
 - logical diagram, 185*
 - virtual service block design, 186*
 - network fabric, 3
 - physical hardware, 1
 - SDN, 224-225
- service blocks
 - deployment, 188*
 - placement, 186*
- services, 3
- storage, 3
- utilization, 2
- virtualization, 2-5
 - conceptual architecture, 5-6*
 - evolution, 5*
 - network, 12-14*
 - server, 6-8*
 - service, 15-16*
 - storage, 9-12*
- data plane (CSR 1000V), 103-104**
 - architecture, 100-102
 - LISP, 171
 - Netmap I/O, 104-105
 - packet flow, 106
 - device initialization, 106-107*
 - Rx, 108*
 - Tx, 107*
 - unicast traffic, 109*
- data synchronization, CSR 1000V**
 - cloudburst, 191**
- Database Abstraction Layer (DAL), 202**
- DCUI (Direct Console User Interface)**
 - processes, VMkernel, 77
- dead queue (IOS scheduler), 65**
- debugging packet loss, 276**
 - CPU usage, 281-282
 - ESXi, 289-292
 - hardware and software configurations, 278-279
 - high-level packet flow, 276-277
 - interface-to-NIC mapping, 281
 - L2 MTU, 280
 - memory usage, 282-288
 - throughput license, 277-278
 - vSwitch packet drops, 289
- define command (virsh), 89**
- deploying OpenStack, 225-233**
 - CSR tenant, 235-242
- deployment models, cloud, 20-21**
- deployment requirements, CSR 1000V**
 - routers, 45-47
- descriptors, VNF, definition, 261-264**
- design**
 - CSR 1000V, 95-98
 - branch design, 162-168*

- hypervisors
 - core partitioning*, 75
 - microkernel architecture*, 74
 - monolithic architecture*, 74
 - operating systems, 60
 - physical resource management*, 60-62
 - software access to resources*, 62
 - design, cloud, 21-23
 - on-demand service, 21
 - destroy command (virsh), 89
 - detach-device command (virsh), 90
 - detach-disk command (virsh), 90
 - detach-interface command (virsh), 90
 - device-based network virtualization, 14-15
 - device-based storage virtualization, 11
 - device drivers
 - ESXi hypervisor, 78-79
 - legacy versus native, 79
 - device initialization packet flow, CSR 1000V data plane, 106-107
 - direct access model, CSR 1000V cloudburst, 191-192
 - Direct Connect, VPC, 214
 - Disabled drop type (IOS), 285
 - disaster recovery using cloudburst, redirection access model, 193
 - DMVPNs (Dynamic Multipoint VPNs), 144-145
 - CSR 1000V, 53
 - overlays, 190
 - DNS (domain name service), diagram, 168
 - Domain 0 (Xen), 93
 - Domain 10 framework
 - abstraction and virtualization, 23
 - automation and orchestration, 23
 - customer interface, 24
 - infrastructure and environment, 22-23
 - organization, governance, and process, 25-26
 - platform and application, 24
 - security and compliance, 24-25
 - service catalog and financials, 24
 - domain controllers, Glance, 202
 - Domain U (XEN), 93
 - Domain U PV (XEN), 93
 - domains, 201
 - dombkstat command (virsh), 90
 - domid command (virsh), 89
 - domifstat command (virsh), 90
 - dominfo command (virsh), 90
 - domname command (virsh), 90
 - domstate command (virsh), 90
 - domuuid command (virsh), 89
 - DR (disaster recovery) using cloudburst, redirection access model, 193
 - Driver, CSR data plane, 102
 - dumpxml command (virsh), 89
 - Dynamic Multipoint VPN (DMVPN), 53, 144-145
-
- ## E
- EC2 (Elastic Compute Cloud), Amazon, 211
 - EEM (Embedded Event Manager), CSR 1000V, 54
 - Elastic Compute Cloud (EC2), Amazon, 211
 - elastic IP, VPC, 213
 - elastic performance, CSR 1000V routers, 47-48
 - elasticity, cloud, 21
 - embedded service processors (ESPs), ASR (Aggregation Service Router), 42-43
 - encapsulation, GRE (Generic Routing Encapsulation), 142
 - enlightened guests, 8
 - Enterprise cloud
 - adoption challenges, 29-30
 - connectivity, 26-28
 - ESPs (embedded service processors)
 - ASR (Aggregation Service Router), 42-43
 - processes, 98
 - ESXi
 - bringing up VM with CSR 1000V, 110-125
 - mapping, 281
 - NIC stats, 290
 - opening screen, 80
 - packet debugging, 289-292
 - port stats, 292
 - VM list, 289

ESXi hypervisor, 75

device drivers, 78-79
 file systems, 79-80
 management, 80-81
 VMkernel, 75-76

CIM processes, 77-78

CPU scheduler, 76

DCUI processes, 77

memory management, 76

VMM processes, 77

VMX processes, 77

ETR (egress tunnel router), 169-170**examples**

Changing the Speed of the Interface, 279

Configuration Script Sample, 240

Control-Processor CLI Output, 282

CSR 1000V as a DMVPN Hub, 149-150

CSR as a Remote Access VPN Server with
 an AnyConnect Client, 153-155

ESXi NIC Stats, 290

ESXi Port Stats, 292

Example of Using the glance CLI to Add
 a CSR 1000V Image, 236

How to Check Throughput Levels and
 License Details, 277

Interface Controller Output, 283-285

Interface show Command, 280

ISR as a DMVPN Spoke, 150-152

Kernel Images Available to GRUB Are
 Listed in menu.lst, 67-68

LMGW Configuration, 177-179

MS/MR Configuration, 179-181

Programs That Are Executed on the Full-
 Multiuser Run Level, 69

QFP Feature Debugging Options,
 286-288

R1 Configuration, 159

R2 Configuration, 159

R3 Configuration, 161

R4 Configuration, 161

R5-1 Configuration, 160

R5-2 Configuration, 160

RR1 Configuration, 158

RR2 Configuration, 158

RR-3 Configuration, 160

RR-4 Configuration, 161

Sample Answer File for Packstack,
 231-233

Sample Definition of VNF Descriptors,
 261-264

Sample of Installed NED Verification, 260

Sample NSO Initiation to Understand
 Input to Be Used in Service and
 Device Model Framework, 253-260

Sample VNF Instantiation, 264

show Commands, 182

show interfaces CLI Output, 283

Snapshot of BGP Update at R4, 162

sw-nic CLI Output, 288

VM List on an ESXi Host, 289

**Extensible Messaging and Presence
 Protocol (XMPP), 32****F**

facilities, data centers, 3

FIFO (first in, first out) scheduling
 algorithm, 60

file systems, VMFS, 79-80

firewalls, ZBFW (Zone Based Firewall), 56

first in, first out (FIFO) scheduling
 algorithm, 60

Forwarding Manager (IOSd), 41

full server virtualization, 8

functions

malloc(), 69

vPATH, 270

G

Generic Routing Encapsulation (GRE)
 tunnels, 142

GET VPNs (Group Encrypted Transport
 VPN), 145-147

Glance component (OpenStack), 34

Glance project, OpenStack, 201-202

GM (Group Member) device, 146

GNU Grand Unified Boot Loader (GRUB),
 66-68

GRE (Generic Routing Encapsulation)
 tunnels, 142

Group Encrypted Transport VPNs (GET
 VPNs), 145-147

Group Member (GM) devices, 146

groups, 201

GRUB (GNU Grand Unified Boot Loader),
 66-68

guest emulator (QEMU), 85-87
 guest mode, Linux kernel, 83

H

hair pinning of traffic, 27
 hardware, speed configuration, 278-279
 hardware hub VPN access, 27
 hardware VPN access, 26
 Heat project, OpenStack, 205
 help command (virsh), 89
 high-level packet flow, debugging, 276-277
 high priority, ready queue, 65
 High queue (IOS scheduler), 38
 Horizon component (OpenStack), 34
 Horizon project, OpenStack, 205
 host-based storage virtualization, 11
 host machines, 272
 hosts, configurations, 275-276
 hybrid cloud, 20
 hybrid kernels, 64
 Hyper-V, 91-92
 hypercalls, 72-73
 hyperthreading setting, BIOS, 276
 hypervisors, 59, 71-72, 94
 design, 74
 core partitioning, 75
 microkernel architecture, 74
 monolithic architecture, 74
 ESXi, 75
 device drivers, 78-79
 file systems, 79-80
 management, 80-81
 VMkernel, 75-78
 Hyper-V, 91-92
 KVM, 82-83
 architectural components, 84-85
 guest emulator, 85-87
 installing CSR 1000V on, 126-137
 Libvirt, 88-91
 PCI passthrough mode, 274
 software, 272
 versus operating systems, 72-73
 VMware, installing CSR 1000V on, 110-125
 Xen, 92-93

IaaS (Infrastructure as a Service), 2, 18-19
 idle queue (IOS scheduler), 65
 Infrastructure as a Service (IaaS). *See* IaaS (Infrastructure as a Service)
 initiation process, CSR 1000V, 99-100
 installation
 CSR 1000V on KVM hypervisor, 126-137
 CSR 1000V on VMware hypervisor, 110-125
 instantiation (VNF), 264-265
 inter-cloud fabric, CSR 1000V cloudburst, 194-195
 interface, changing speed, 278-279
 Interface Manager (IOSd), 41
 interface-to-NIC mapping, configuration, 281
 Internet for transport, Enterprise cloud connectivity, 26-28
 Internet gateway (VPC), 213
 Internetworking Operating System (IOS). *See* IOS (Internetworking Operating System)
 I/O configuration, CSR 1000V, troubleshooting, 272-276
 I/O stack, Netmap, 105
 IOS (Internetworking Operating System), 37-39
 boot process, 66-67
 IOSd (IOS daemon), 40-41
 kernel, 64
 scheduler, 65-66
 scheduler, 37
 XE architecture, 39
 kernel, 40
 XE drop types, 285
 IOSd (IOS daemon), 95
 IOS XE, 96
 versus IOS, 96-98
 IOS XE operating system, 95
 IP mobility, LISP, 175
 IP SLA (IP Service Level Agreement), CSR 1000V, 54
 IPsec VPNs, 142
 with crypto maps, 143-144
 offloading, 3

Ipsilon Networks, 13
 Ipv4NoAdj drop type (IOS), 286
 Ipv4NoRoute drop type (IOS), 286
 IPv6 migration, LISP, 175
 ITR (ingress tunnel router), 169-170

K

Kernel-based Virtual Machine (KVM). *See*
 KVM (Kernel-based Virtual Machine)
 kernels, 63
 ESXi hypervisor, VMkernel, 75-78
 hybrid, 64
 IOS, 64
 memory manager, 65-66
 scheduler, 65
 IOS XE, 40
 KVM (Kernel-based Virtual Machine),
 82-83
 architectural components, 84-85
 guest emulator, 85-87
 Linux, memory management, 69-71
 microkernels, 63
 hypervisor architecture, 74

Key Server (KS), 146
 Keystone component (OpenStack), 34
 Keystone project, OpenStack, 199-201
 KS (Key Server), 146
 KVM (Kernel-based Virtual Machine),
 82-83
 architectural components, 84-85
 guest emulator, 85-87
 hypervisor
 installing CSR 1000V on, 126-137
 Libvirt, 88-91

L

L2 MTU, configuration, 280
 L2VPNs (Layer 2 VPNs), 141
 L3VPNs (Layer 3 VPNs), 141-143
 DMVPNs (Dynamic Multipoint VPNs),
 144-145
 GET VPNs (Group Encrypted Transport
 VPN), 145-147
 GRE tunnels, 142
 IPsec VPNs, 142
 IPsec VPNs with crypto maps, 143-144

MPLS VPNs, 142
 site-to-site VPNs, 143
 SSL VPNs, 147-148
 legacy drivers versus native drivers, 79

Libvirt

 management daemon, 88
 user tools, 89-91
 virsh, 89-91

Linux

 memory management, 69
 caching, 71
 overcommitment, 69-70
 swap space, 69-71
 versus Netmap I/O, 105

LISP (Location/ID Separation Protocol),

 168-169, 175, 190
 control plane, 171-175
 CSR 1000V, 54
 data plane, 171
 ETR (egress tunnel router), 169-170
 IP mobility, 175
 IPv6 migration, 175
 ITR (ingress tunnel router), 169-170
 MR (MAP Resolver), 170
 network-to-network connectivity,
 175-176
 network-to-network interconnection
 configuration, 176, 179-182
 packet header, 172
 PETR (proxy egress tunnel router),
 170-171
 PITR (proxy ingress tunnel router), 170

LISP-to-MPLS Gateway (LMGW),

 176, 179
 list command (virsh), 89
 LMGW (LISP-to-MPLS Gateway),
 176, 179

Locator/ID Separation Protocol (LISP).

See LISP (Locator/ID Separation
 Protocol)

low priority, ready queue, 65
 Low queue (IOS scheduler), 38

M

mainframes, virtualization, 17
 malloc() function, 69
 management, automation, 247

management daemon (Libvirt), 88
 user tools, 89-91

map resolver (MR), 176

map server (MS), 176

MBR (Master Boot Record), 67

measured services, cloud, 21

medium priority, ready queue, 65

Medium queue (IOS scheduler), 38

memory, physical, 76

memory management

- Linux
 - caching*, 71
 - over commitment*, 69-70
 - swap space*, 69-71
- VMkernel, 76

memory manager (IOS), 65-66

memory usage, configuration, 282-288

microkernel architecture, hypervisors, 74

microkernels, 63

migrate command (virsh), 90

modes of operation (KVM), 83

monolithic architecture, hypervisors, 74

MPLS (Multiprotocol Label Switching), 14

- VPNs (virtual private networks), 54-55, 142

MPLS over GRE, 190

MR (MAP Resolver), 170

MTUs (maximum transmission units)

- L2, configuration, 280

multipoint L2VPNs, 141

Multiprotocol Label Switching (MPLS). *See* MPLS (Multiprotocol Label Switching)

multitenancy, 21, 185

multitenant data center, 16-18

- CSR, 185-190
 - zone connectivity*, 188
- logical diagram, 185
- virtual service block design, 186

N

NAS (network-attached storage), 3

native drivers versus legacy drivers, 79

NBAR2 (Network Based Application Recognition), 52

NEDs, installation verification, 260-261

NETCONF, 166-168

Netmap I/O

- CSR 1000V data plane, 104-105
- versus Linux, 105

network-attached storage (NAS), 3

Network Based Application Recognition (NBAR2), 52

network-based storage virtualization, 11

Network Configuration Protocol (NetConf), 32

network connectivity, CSR 1000V

- cloudburst, 190

network data centers, virtualization, 15

network extension from premises

- deployment, CSR 1000V routers, 51

network fabric, data centers, 3

Network Functions Virtualization (NFV), 15, 33, 156, 223

network-level hypervisor, virtualization, 15

network-to-network connectivity, LISP, 175-176

network-to-network interconnection

- configuration, LISP, 176-182

network virtualization, 12-13

- device-based, 14-15
- evolution, 13-14
- protocol-based, 14

networking

- NFV (network function virtualization), 33
- OpenStack, 34
- SDN (software-defined networking), 30-31
 - ONF (Open Networking Foundation)*, 31-32
 - OpenDaylight*, 32-33

networks

- orchestrating solutions, 247
- VPNs (virtual private networks), 141
 - DMVPNs (Dynamic Multipoint VPNs)*, 144-145
 - GET VPNs (Group Encrypted Transport VPN)*, 145-147
 - IPsec*, 142
 - IPsec with crypto maps*, 143-144
 - L2VPNs*, 141
 - L3VPNs*, 142-148
 - remote access into cloud*, 153-155
 - service gateway*, 148-153
 - site-to-site*, 143
 - SSL VPNs (Secure Sockets Layer VPN)*, 147-148

Neutron component (OpenStack), 34
Neutron plugin, instantiating CSR to OpenStack, 242-245
Neutron project, OpenStack, 202-203
Neutron routers, CSR 1000V, 206-209
NFV (Network Functions Virtualization), 15, 156, 223
 NSO orchestration with OpenStack, 252-253, 260-266
NFV MANO, 268
NFV Orchestrator, 268
Nova component (OpenStack), 34
Nova project, OpenStack, 198-199
NSO tool, 249-251
 initiation in service and device model framework, 253, 260
 installed NED verification, 260-261
 NFV orchestration with OpenStack, 252-253, 260-261, 264-266
 VNF (virtual network function)
 descriptor definition, 261, 264
 instantiation, 264-265
NVF (network function virtualization), 33

O

on-demand service, cloud, 21
ONF (Open Networking Foundation), 31-32
OpenDaylight, 32-33
OpenFlow protocol, 31-32
OpenStack, 34, 196, 225
 Create Network submenu, 237-238
 Create Router dialog, 238
 CSR, instantiating Neutron plugin to, 242-245
 dashboard login, 235
 deploying, 225-233
 CSR tenant, 235-242
 network subnet menu, 238
 private cloud deployment, CSR 1000V, 195-211
 projects, 197
 Ceilometer, 205
 Cinder, 204
 Glance, 201-202
 Heat, 205
 Horizon, 205

Keystone, 199-201
 Neutron, 202-203
 Nova, 198-199
 Sabara, 205
 Swift, 205
 Trove, 205
 tenants, 265-266
 VM image creation, 235
operating systems, 59
 boot process, 66-67
 design, 60
 physical resource management, 60-62
 software access to resources, 62
IOS XE, 95-96
 versus IOS, 96-98
kernels, 63
 hybrid, 64
 IOS, 64-66
 microkernels, 63
Linux, memory management, 69-71
 shared resource access, 72
 versus hypervisors, 72-73
 virtualization, 8
orchestration, 267
 network solutions, 247
 NFV MANO, 268
 PNSC (Prime Network Services Controller), 269-270
 versus automation, 247
 VMS (Virtual Managed Services), 267-268
OTV (Overlay Transport Virtualization), 190
 CSR 1000V, 55
overcommitment, memory, Linux, 69-70
Overlay Transport Virtualization (OTV). See OTV (Overlay Transport Virtualization)
overlays, SDN, 224

P

PaaS (Platform as a Service), 2, 18-19
packet flow, CSR 1000V data plane, 103
 device initialization, 106-107
 Netmap I/O, 104-105
 Rx, 108
 Tx, 107
 unicast traffic, 109

- packet loss, debugging, 276
 - CPU usage, 281-282
 - ESXi, 289-292
 - hardware and software configurations, 278-279
 - high-level packet flow, 276-277
 - interface-to-NIC mapping, 281
 - L2 MTU, 280
 - memory usage, 282-288
 - throughput license, 277-278
 - vSwitch packet drops, 289
- packets, LISP, 172
- Packstack, deploying OpenStack, 225-233
- paged memory, 61
- para-virtualization, server, 8
- partitioning, core, hypervisors, 75
- pay-as-you-use service, 21
- PCI passthrough mode, hypervisor software, 274
- Performance Routing (PfR), 55
- performance tuning, CSR 1000V, 137-139
- PETR (proxy egress tunnel router), 170-171
- PfR (Performance Routing), 55
- physical infrastructure, network virtualization, 15
- physical memory, 76
- physical resource management, operating systems, 60-62
- pinning, CPUs, 138
- PITR (proxy ingress tunnel router), 170
- Platform as a Service (PaaS). *See* PaaS (Platform as a Service)
- Platform Manager (IOSd), 41
- PNSC (Prime Network Services Controller), 248
 - orchestration, 269-270
- point-to-point L2VPNs, 141
- policy IPsec VPNs, 143
- pool manager (memory manager), 65
- Power setting, BIOS, 276
- preemption scheduling algorithm, 61
- Prime Network Services Controller (PNSC). *See* PNSC (Prime Network Services Controller)
- priority scheduling algorithm, 61
- private cloud, 20
- private cloud deployment in OpenStack, CSR 1000V, 195-211
- processes
 - ESP, 98
 - RP, 97
 - SIP/SPA, 98
 - VMkernel, 77-78
- programmable fabric, SDN, 224
- projects, OpenStack, 197
 - Ceilometer, 205
 - Cinder, 204
 - Glance, 201-202
 - Heat, 205
 - Horizon, 205
 - Keystone, 199-201
 - Neutron, 202-203
 - Nova, 198-199
 - Sahara, 205
 - Swift, 205
 - Trove, 205
- protocol-based network protocol, 14
- protocols
 - LISP (Location/ID Separation Protocol), 54, 168-169, 175
 - control plane*, 171-175
 - data plane*, 171
 - ETR (egress tunnel router)*, 169-170
 - IP mobility*, 175
 - IPv6 migration*, 175
 - ITR (ingress tunnel router)*, 169-170
 - MR (MAP Resolver)*, 170
 - network-to-network connectivity*, 175-176
 - network-to-network interconnection configuration*, 176, 179-182
 - PETR (proxy egress tunnel router)*, 170-171
 - PITR (proxy ingress tunnel router)*, 170
 - OpenFlow, 31-32
- provisioning, automation, 247
- public cloud, 20
- public cloud deployment
 - CSR, Amazon web services, 211-215
 - CSR 1000V, 211
 - Amazon web services*, 216-222

Q

QEMU (Quick Emulator)

- architectural components, 84-85
- guest emulator, 85-87
- KVM architecture, 87

QFP (QuantumFlow Processor), 43

- debugging options, 286-288

QFP uCode (packet processing), CSR data plane, 102

quit command (virsh), 90

R

Radio Aware Routing, CSR 1000V, 56

rapid deployment, CSR 1000V routers, 49

ready queue (IOS scheduler), 65

reboot command (virsh), 90

redirection access model, CSR 1000V
cloudburst, 192-193

Redundancy Group Infrastructure, CSR
1000V, 56

region manager (memory manager), 65

regions, VPC, 214

resource pooling, 21

REST API, 202

restore command (virsh), 90

resume command (virsh), 90

ring manipulation, Netmap I/O, 105

RLOC (Routing Locator), 170

round-robin scheduling algorithm, 60

route reflector (RR), 176

routers. *See also* CSR (cloud service router)

ASR 1000, 41

*ESPs (embedded service processors),
42-43*

RP (route processor), 42

SIP (SPA interface processor), 43

CSR (cloud service router)

BPG route reflector, 155-162

host configuration, 275-276

hypervisors, 59

*LISP (Locator/ID Separation
Protocol), 168-182*

OpenStack, 235-242

*public cloud deployment, Amazon
web services, 211-215*

*as Remote Access VPN server,
153-155*

*remote VPN access into Cloud,
153-155*

secure inter-cloud connectivity, 152

CSR 1000V, 44-45

deployment requirements, 45-47

*elastic performance and scaling,
47-48*

*network extension from premises to
cloud, 51*

rapid deployment, 49

routing flexibility, 49

secure cloud VPN gateway, 50-51

segmentation within cloud, 52

ETR (Egress Tunnel Router), 169-170

ITR (Ingress Tunnel Router), 169-170

Neutron, CSR 1000V, 206-209

PETR, 170-171

PITR, 170

software, 16

tenant, CSR 1000V, 209-211

routing

flexibility, 49

PfR (Performance Routing), 55

Radio Aware Routing, 56

tables, 213

RP (route processor)

ASR (Aggregation Service Router), 42

processes, 97

Rx packet flow, CSR 1000V data
plane, 108

S

S3 (Simple Storage Service), Amazon, 212

SaaS (Software as a Service), 2, 18-20

Sahara project, OpenStack, 205

SAN (storage-area network), 3

save command (virsh), 90

scaling CSR 1000V routers, 47-48

scheduler (IOS), 37, 65

scheduling algorithms, 60-62

SDN (software-defined networking), 1,
30-31, 223

abstract layer creation, 223

application-centric infrastructure, 224

data center, 224-225

- framework, 223
- ONF (Open Networking Foundation), 31-32
- OpenDaylight, 32-33
- overlays, 224
- programmable fabric, 224
- secure cloud VPN gateway deployment, CSR 1000V routers, 50-51
- secure inter-cloud connectivity, CSR, 152
- Secure Sockets Layer (SSL), 3
 - VPNs (SSL VPNs), 147-148
- security groups, VPC, 213
- segmentation within cloud, CSR 1000V routers, 52
- segmented memory allocation, 61
- server virtualization, 6-7
 - full, 8
 - OS (operating system), 8
 - para-, 8
 - storage, 10
- service blocks, data centers
 - deployment, 188
 - placement, 186
- service virtualization, 15-16
- services
 - data centers, 3
 - IaaS (Infrastructure as a Service), 18-19
 - PaaS (Platform as a Service), 19
 - SaaS (Software as a Service), 20
- Setmaxmem command (virsh), 90
- Setmem command (virsh), 90
- Setvcpus command (virsh), 90
- shared resources, access, operating systems, 72
- Shortest Process Next (SPN) scheduling algorithm, 60
- Shortest Remaining Time (SRT) algorithm, 61
- show commands, 181-182
- show interface command, 278-279
- show interface controller command, 283
- show interfaces command, 282
- shutdown command (virsh), 90
- Simple Storage Service (S3), Amazon, 212
- Single Root I/O Virtualization (SR-IOV), 274-275
- SIP (SPA interface processor), ASR (Aggregation Service Router), 43
- SIP/SPA processes, 98
- site-to-site VPNs, 143
- SNIA (Storage Networking Industry Association), 10
- sockets settings, hosts, 275
- software
 - access to physical resources, operating systems, 62
 - speed configuration, 278-279
- Software as a Service (SaaS). *See* SaaS (Software as a Service)
- software crypto engine, CSR 1000V, 103
- software-defined networking (SDN). *See* SDN (software-defined networking)
- software router, 16
- software VPN access, 27
- SPA interface processor (SIP), 43
- speed command, 278
- SpeedStep setting, BIOS, 276
- SPN (Shortest Process Next) scheduling algorithm, 60
- SR-IOV (Single Root I/O Virtualization), 274-275
- SRT (Shortest Remaining Time) algorithm, 61
- SSL (Secure Sockets Layer), 3
 - VPNs (Secure Sockets Layer VPNs), 147-148
- start command (virsh), 89
- static routing VPNs, 143
- statistics drop command, 285
- storage, 3
- storage-area network (SAN), 3
- Storage Networking Industry Association (SNIA), 10
- storage virtualization, 9-11
 - device-based, 11
 - hidden complexity, 12
 - host-based, 11
 - network-based, 11
 - performance, 12
 - thin provisioning, 12
- subnets, VPC, 212
- supervisors, 72
- suspend command (virsh), 90
- swap caches, 71
- swap files, 71

swapping memory, Linux, 69-71
 Swift component (OpenStack), 34
 Swift project, OpenStack, 205
 switches, vSwitch, 272-273
 packet drops, 289
 synchronization, Netmap, 105
 system design, CSR 1000V, 95-98

T

Tail-f framework, 166
 Tail-f tool, 249, 251
 NFV orchestration with OpenStack,
 252-253, 260-261, 264-266
 TailDrop drop type (IOS), 286
 tenant deployment, OpenStack, CSR,
 235-240, 242
 tenant routers
 CSR 1000V, 209-211
 tenants, 201
 OpenStack, 265-266
 thin client mode (SSL VPNs), 147
 thrashing, 71
 throughput, appropriate license, 277-278
 Tiny Code Generator (TCG), 87
 traffic, hair pinning, 27
 troubleshooting, CSR 1000V
 architecture overview, 271-272
 I/O configuration, 272-276
 Trove project, OpenStack, 205
 tunnel mode (SSL VPNs), 147
 Tx packet flow, CSR 1000V data
 plane, 107

U

uCode, CSR data plane, 102
 UnconfiguredIpv4Fia drop type (IOS), 286
 UnconfiguredIpv6Fia drop type (IOS), 286
 undefine command (virsh), 90
 unicast traffic packet flow, CSR 1000V
 data plane, 109
 UNIVAC-I mainframe computer, 1
 unstructured data, 4
 user tools, Libvit, 89-91
 utilization, data centers, 2

V

vCenter, 81
 Vcpuinfo command (virsh), 90
 Vcpupin command (virsh), 90
 VIM (Virtualized Infrastructure
 Manager), 268
 virsh command-line tool, 89-91
 Virtual Extensible LAN (VxLAN), 56, 190
 Virtual Machine File System (VMFS),
 79-80
 Virtual Machine Manager, 71, 127,
 130-134. *See also* hypervisors
 virtual machines, KVM (Kernel-based
 Virtual Machine), 82-83
 architectural components, 84-85
 guest emulator, 85-87
 Virtual Managed Services (VMS). *See* VMS
 (Virtual Managed Services)
 Virtual Private Cloud (VPC), Amazon, 211
 Virtual Private LAN Services (VPLS), 55
 virtual private networks (VPNs). *See* VPNs
 (virtual private networks)
 virtual routing and forwarding (VRF), 14
 virtual service block design, multitenant
 data centers, 186
 virtual switches, vSwitch, 272-273
 virtualization, 71
 ASR 1001 into CSR 1000V, 98
 branch, 164-168
 data centers, 2-4
 conceptual architecture, 5-6
 evolution, 5
 mainframes, 17
 network, 12-13
 device-based, 14
 evolution, 13-14
 protocol-based, 14
 NFV (Network Functions Virtualization),
 33, 156, 223
 OTV (Overlay Transport Virtualization),
 CSR 1000V, 55
 server, 6-8
 storage, 11
 service, 15-16
 SR-IOV (Single Root I/O Virtualization),
 274-275

- storage, 9-10
 - device-based*, 11
 - hidden complexity*, 12
 - host-based*, 11
 - network-based*, 11
 - thin provisioning*, 12
- Virtualized Infrastructure Manager (VIM)**, 268
- VMs (virtual machines)**
 - bringing up on ESXi with CSR 1000V, 110-125
 - layers, CSR, 103
- VMFS (virtual machine file system)**, 79-80
- VMkernel, ESXi hypervisor**, 75-76
 - CIM processes, 77-78
 - CPU scheduler, 76
 - DCUI processes, 77
 - memory management, 76
 - VMM processes, 77
 - VMX processes, 77
- VMM (virtual machine manager)**. *See* hypervisors
- VMS (Virtual Managed Services)**, 248
 - orchestration, 267-268
- VMware hypervisor, installing CSR 1000V on**, 110-125
- VMX processes, VMkernel**, 77
- VNF**
 - descriptor definition, 261, 264
 - instantiation, 264-265
 - Manager, 268
- vPATH function**, 270
- VPC (Virtual Private Cloud)**, 212
 - Amazon, 211
 - availability zones, 214
 - Direct Connect, 214
 - elastic IP, 213
 - Internet gateway, 213
 - regions, 214
 - routing tables, 213
 - security groups, 213
 - subnets, 212
- VPLS (Virtual Private LAN Services)**, CSR 1000V, 55
- VPNs (virtual private machines)**, 141
 - DMVPNs (Dynamic Multipoint VPNs), 53, 144-145
 - GET VPNs (Group Encrypted Transport VPN), 145-147
 - IPsec, 142
 - IPsec with crypto maps, 143-144
 - L2VPNs, 141
 - L3VPNs, 142-143
 - DMVPNs, 144-145
 - GET VPNs, 145-147
 - GRE tunnels, 142
 - IPsec VPNs, 142-144
 - MPLS VPNs, 142
 - site-to-site VPNs, 143
 - SSL VPNs, 147-148
 - MPLS (Multiprotocol Label Switching), 54
 - remote access into cloud, 153-155
 - service gateway, 148-153
 - site-to-site, 143
 - SSL VPNs (Secure Sockets Layer VPN), 147-148
- VRF (virtual routing and forwarding)**, 14
- vSwitch**, 272-273
 - packet drops, troubleshooting, 289
- VxLAN (Virtual Extensible LAN)**, 190
 - CSR 1000V, 56

W-X

Windows Server Virtualization, 91
 workload migration, CSR 1000V
 cloudburst, 191

XE architecture, IOS, 39
 kernel, 40

Xen hypervisor, 92-93
 XMPP (Extensible Messaging and Presence Protocol), 32

Y-Z

YANG, 166-168

ZBFW (Zone Based Firewall), CSR 1000V,
 56-57

zero-copy model, Netmap, 105
 Zone Based Firewall (ZBFW), 56
 zones, connectivity, multitenant data
 center with CSR 1000V, 188



Connect, Engage, Collaborate

The Award Winning Cisco Support Community

Attend and Participate in Events

Ask the Experts

Live Webcasts

Knowledge Sharing

Documents

Blogs

Videos

Top Contributor Programs

Cisco Designated VIP

Hall of Fame

Spotlight Awards

Multi-Language Support



<https://supportforums.cisco.com>



REGISTER YOUR PRODUCT at [CiscoPress.com/register](https://ciscopress.com/register) Access Additional Benefits and SAVE 35% on Your Next Purchase

- Download available product updates.
- Access bonus material when applicable.
- Receive exclusive offers on new editions and related products.
(Just check the box to hear from us when setting up your account.)
- Get a coupon for 35% for your next purchase, valid for 30 days.
Your code will be available in your Cisco Press cart. (You will also find it in the Manage Codes section of your account page.)

Registration benefits vary by product. Benefits will be listed on your account page under Registered Products.

CiscoPress.com – Learning Solutions for Self-Paced Study, Enterprise, and the Classroom
Cisco Press is the Cisco Systems authorized book publisher of Cisco networking technology, Cisco certification self-study, and Cisco Networking Academy Program materials.

At [CiscoPress.com](https://ciscopress.com) you can

- Shop our books, eBooks, software, and video training.
- Take advantage of our special offers and promotions (ciscopress.com/promotions).
- Sign up for special offers and content newsletters (ciscopress.com/newsletters).
- Read free articles, exam profiles, and blogs by information technology experts.
- Access thousands of free chapters and video lessons.

Connect with Cisco Press – Visit [CiscoPress.com/community](https://ciscopress.com/community)

Learn about Cisco Press community events and programs.



Cisco Press