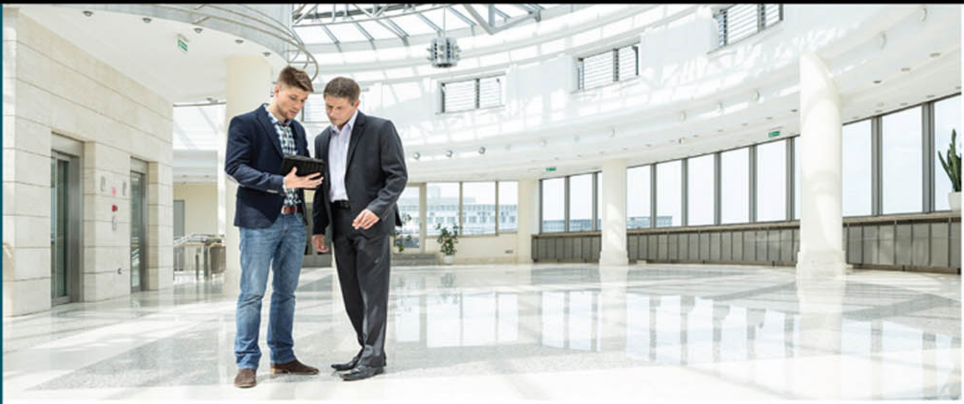


Virtual Routing in the Cloud

Arvind Durai
Stephen Lynn
Amit Srivastava

ciscopress.com

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

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

Published by:
Cisco Press
800 East 96th Street
Indianapolis, IN 46240 USA

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

Trademark Acknowledgments

Special Sales

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

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

Reader Services

Contents

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.

Chapter 4

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.

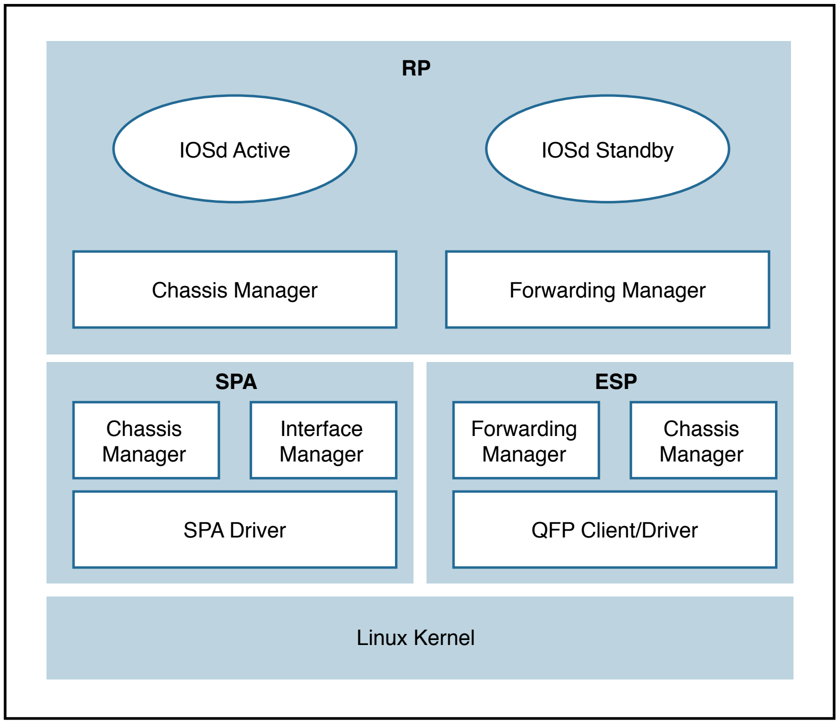


Figure 4-1 *ASR 1001 Platform Logical Architecture*

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.

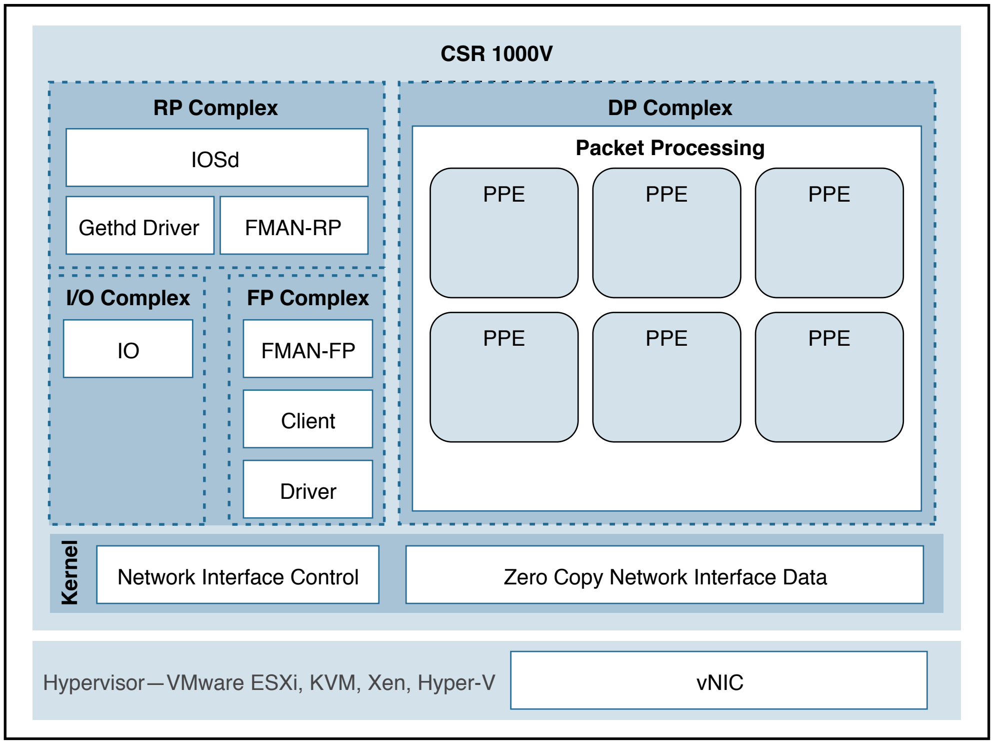


Figure 4-2 *CSR 1000V High-Level Architecture*

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.

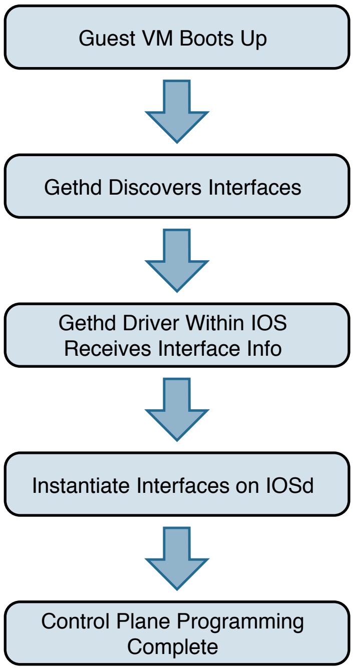

Guest VM Boots Up
Gethd Discovers Interfaces
Gethd Driver Within IOS Receives Interface Info
Instantiate Interfaces on IOSd
Control Plane Programming Complete

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.

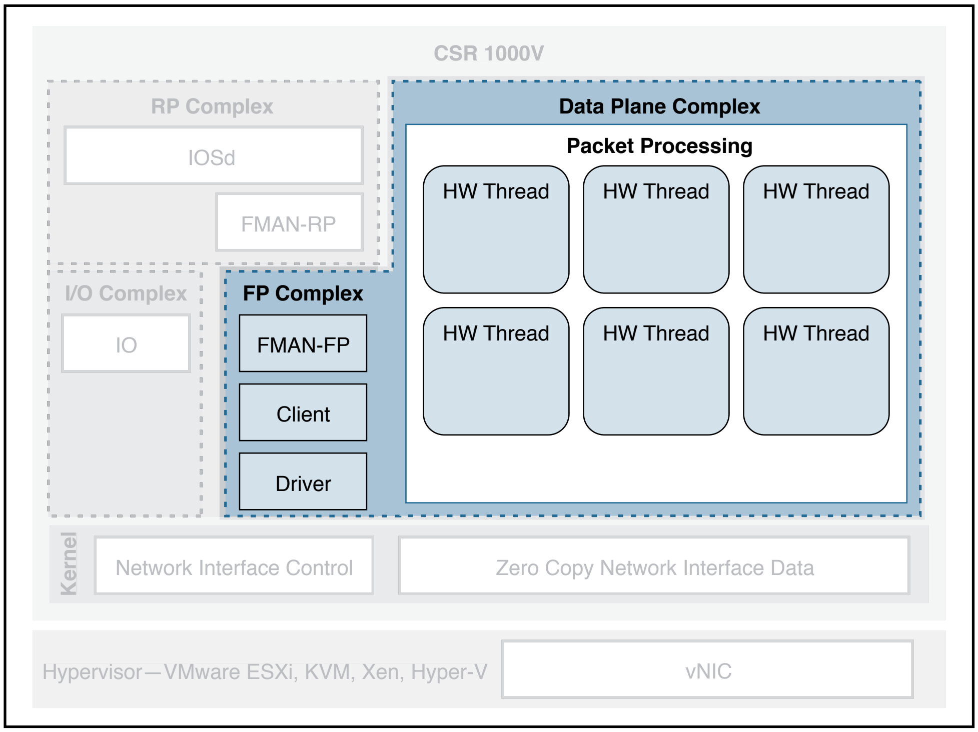


Figure 4-4 *CSR 1000V Data Plane Architecture*

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.

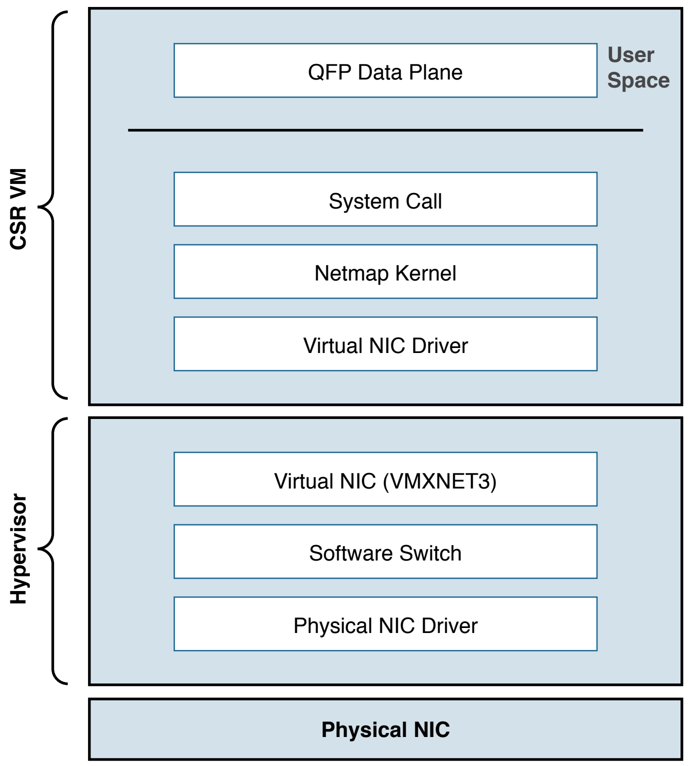


Figure 4-5 *CSR VM Layers*

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.

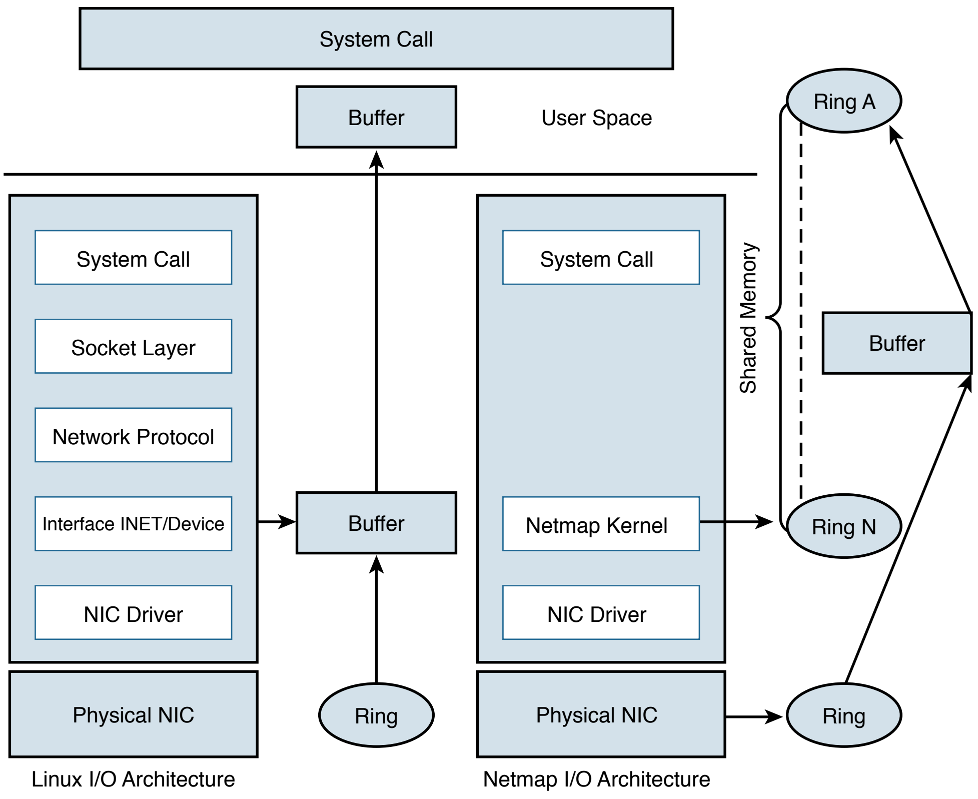


Figure 4-6 *Linux Versus Netmap I/O Comparison*

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.

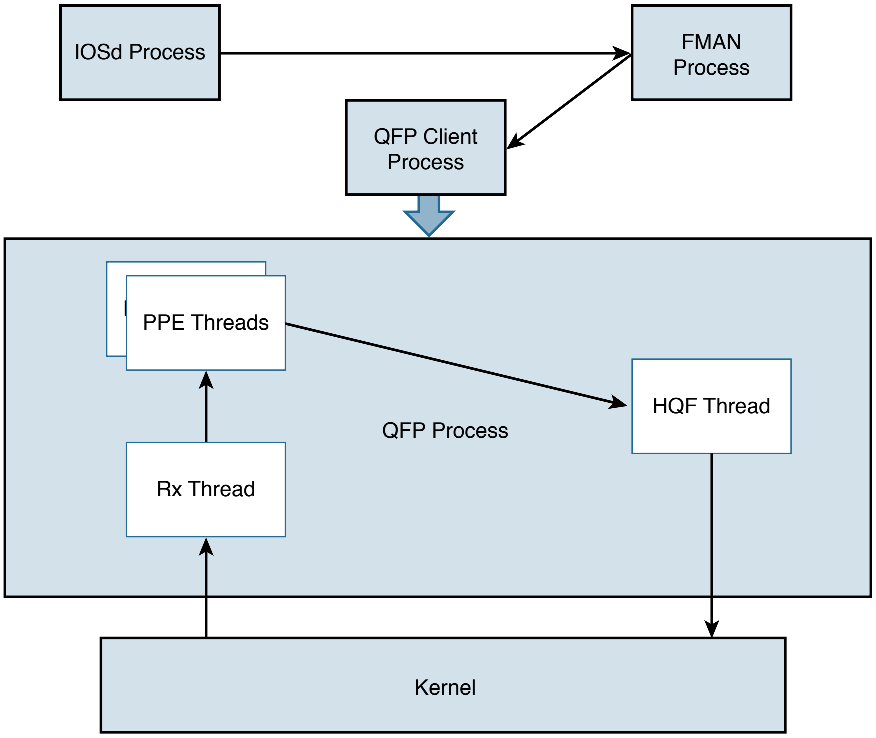


Figure 4-7 *Flowchart for Packet Flow*

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.

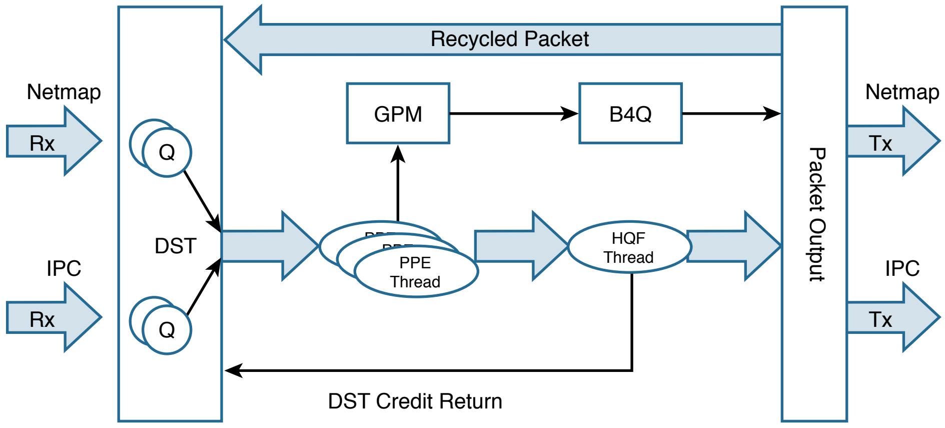


Figure 4-8 *CSR 1000V Packet Flow in the QFP Process*

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."

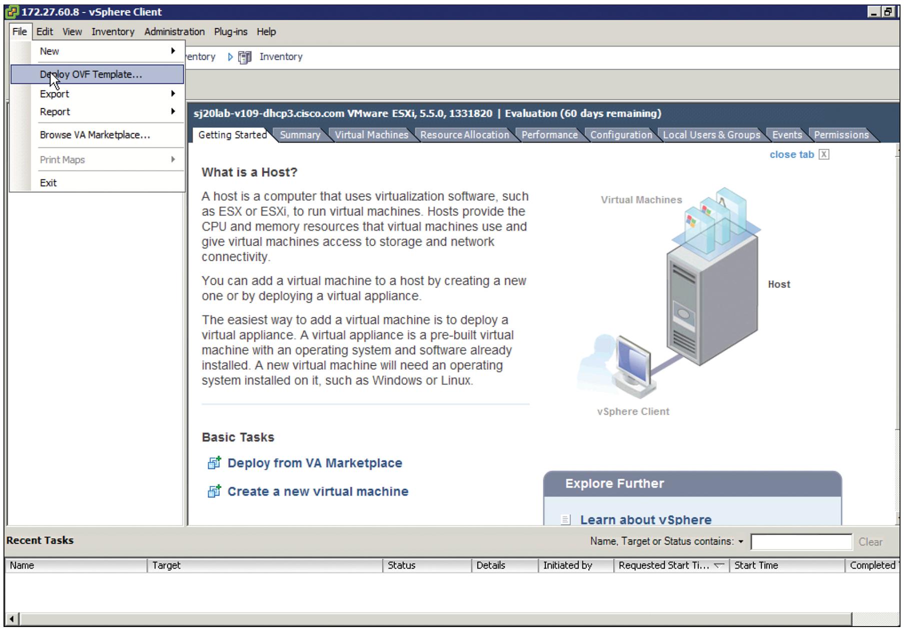
The following steps assume ESXi is already installed. Please refer to the VMware ESXi installation guide for setting up the ESXi if it is not already installed.

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.



What is a Host?

A host is a computer that uses virtualization software, such as ESX or ESXi, to run virtual machines. Hosts provide the CPU and memory resources that virtual machines use and give virtual machines access to storage and network connectivity.

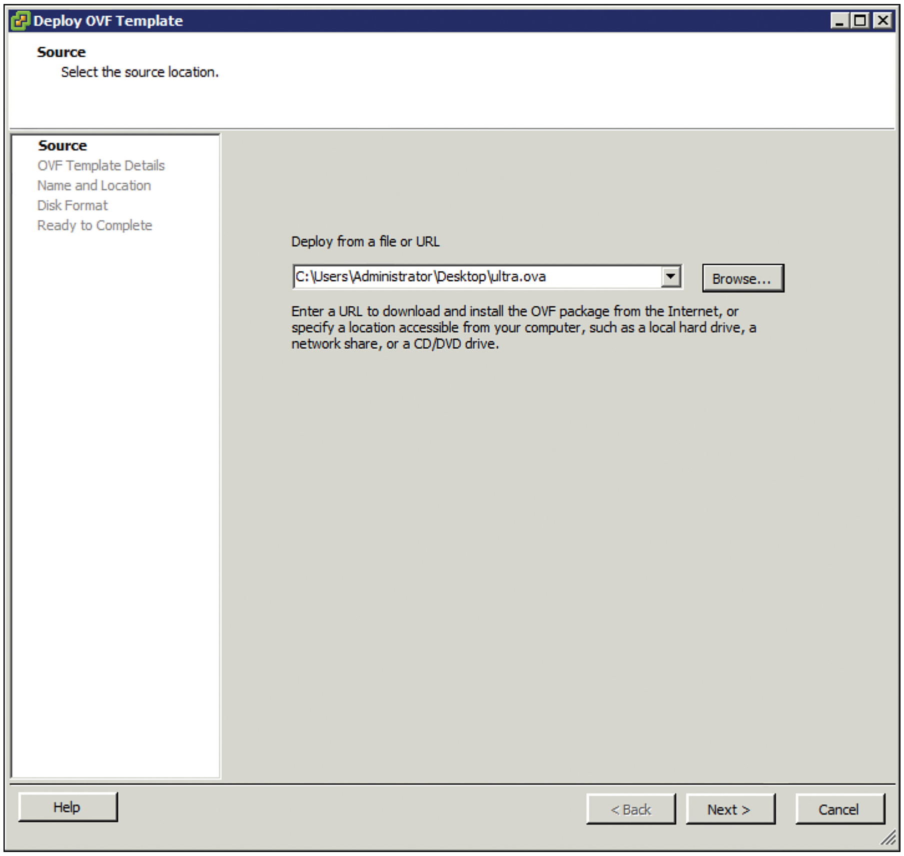
You can add a virtual machine to a host by creating a new one or by deploying a virtual appliance.

The easiest way to add a virtual machine is to deploy a virtual appliance. A virtual appliance is a pre-built virtual machine with an operating system and software already installed. A new virtual machine will need an operating system installed on it, such as Windows or Linux.

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.



Source
Select the source location.

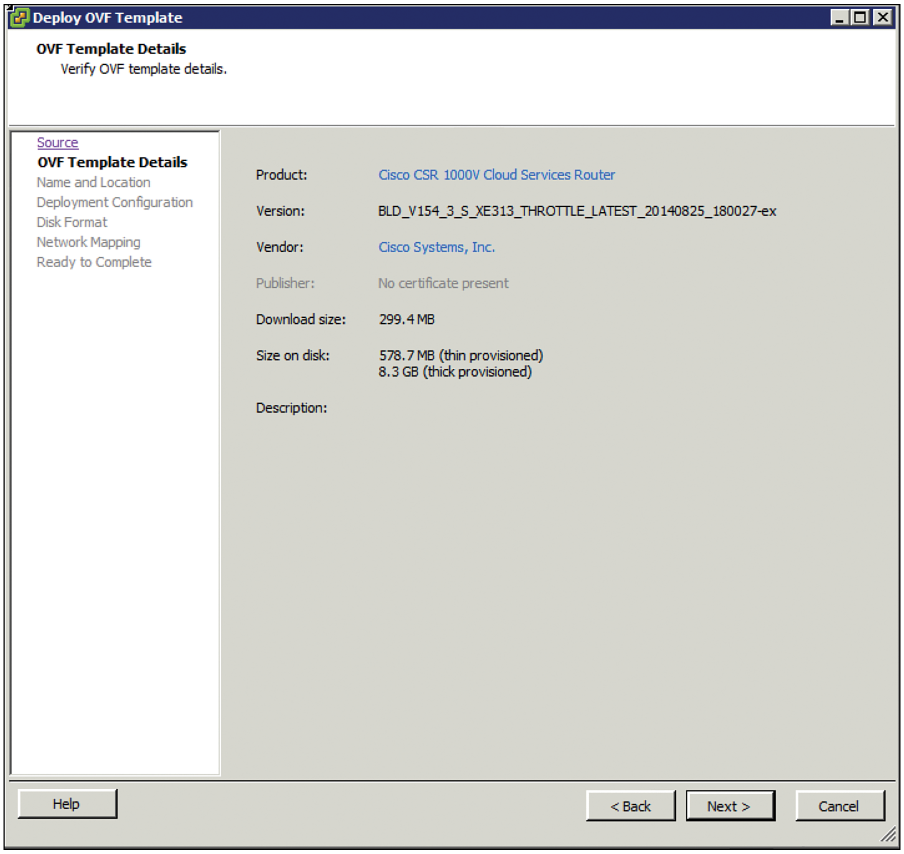
Deploy from a file or URL

C:\Users\Administrator\Desktop\ultra.ova

Enter a URL to download and install the OVF package from the Internet, or specify a location accessible from your computer, such as a local hard drive, a network share, or a CD/DVD drive.

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.



Product: Cisco CSR 1000V Cloud Services Router
Version: BLD_V154_3_S_XE313_THROTTLE_LATEST_20140825_180027-ex
Vendor: Cisco Systems, Inc.
Publisher: No certificate present
Download size: 299.4 MB
Size on disk: 578.7 MB (thin provisioned)
8.3 GB (thick provisioned)
Description:

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.)

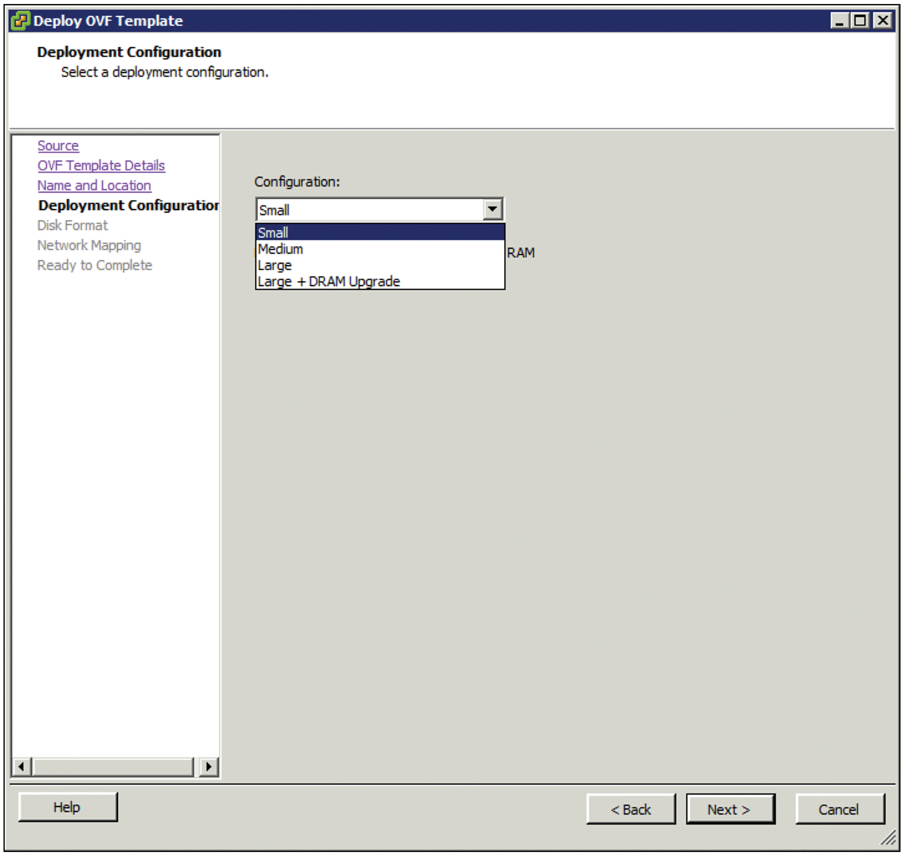


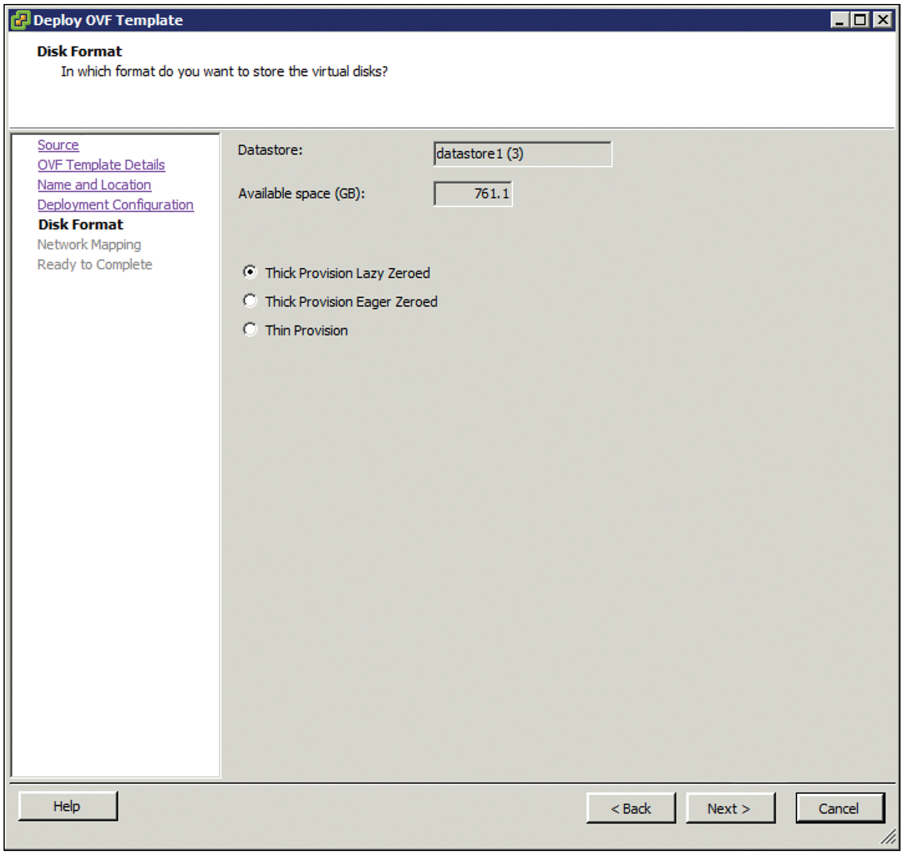
Figure 4-12 *Deploying the OVF Template: Selecting the System Memory Profile for CSR 1000V*

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.



Datastore: datastore1 (3)

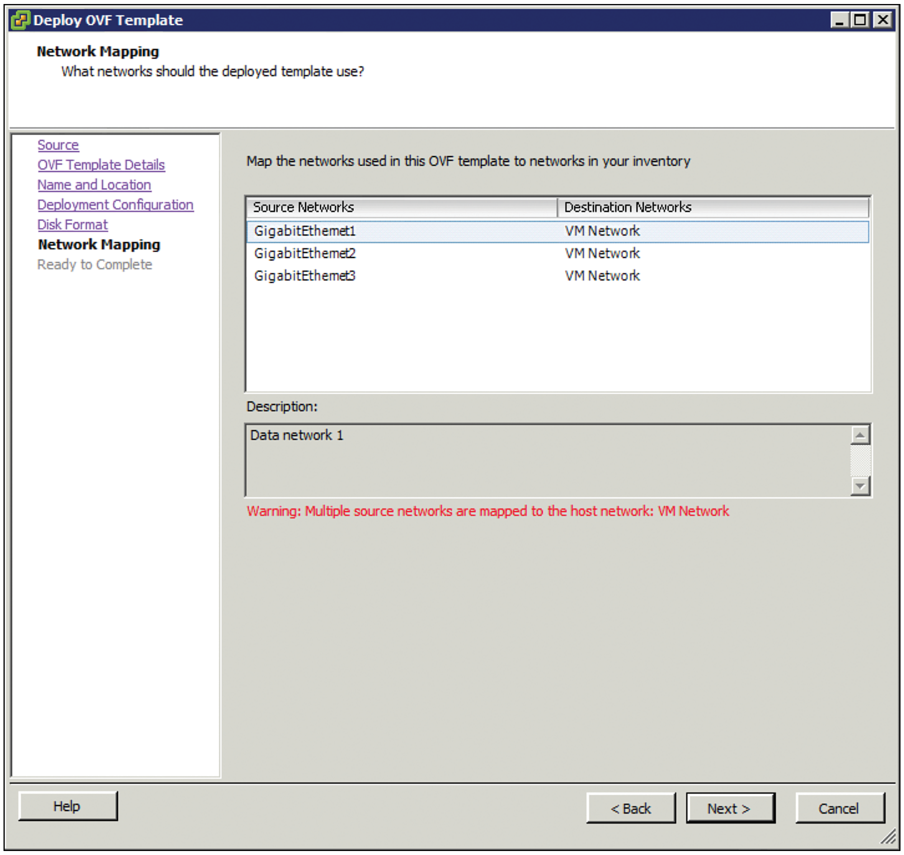
Available space (GB): 761.1

[✓ Thick Provision Lazy Zeroed]
Thick Provision Eager Zeroed
Thin Provision

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.


Deploy OVF Template

Network Mapping

What networks should the deployed template use?

Map the networks used in this OVF template to networks in your inventory

Source Networks	Destination Networks
GigabitEthernet1	VM Network
GigabitEthernet2	VM Network
GigabitEthernet3	VM Network

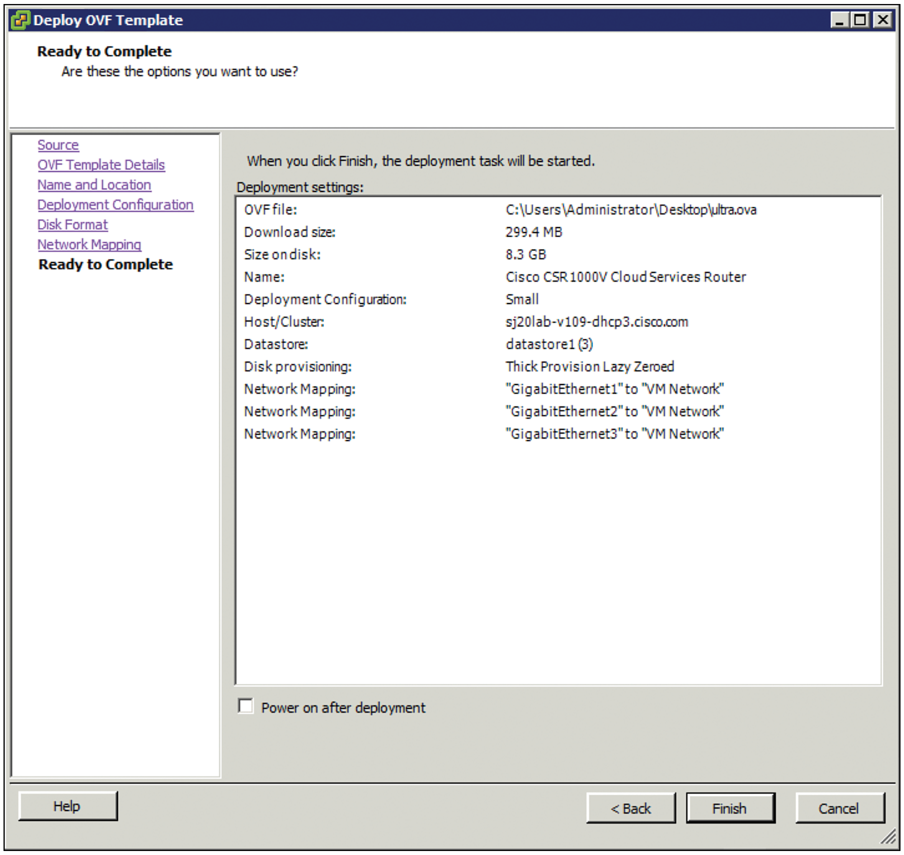
Description:

Data network 1

Warning: Multiple source networks are mapped to the host network: VM Network

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.



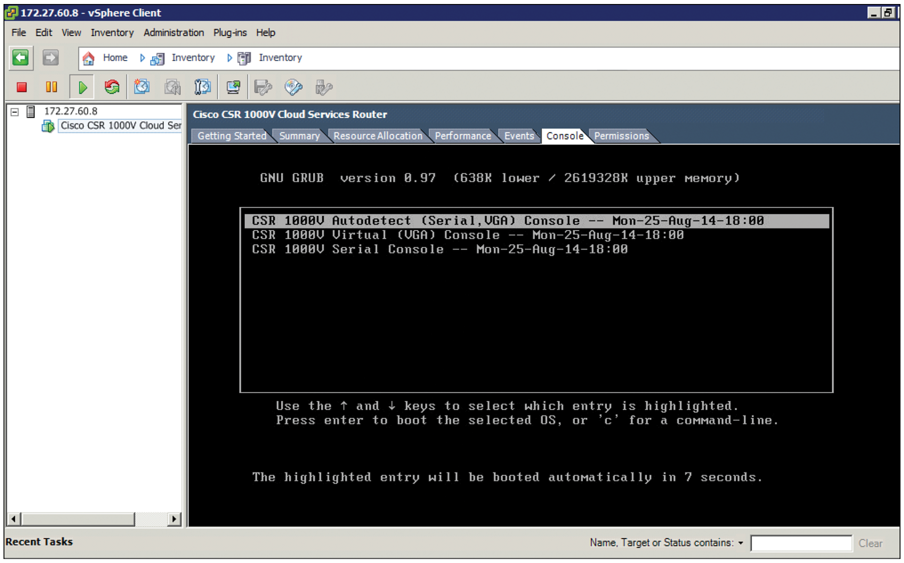
When you click Finish, the deployment task will be started.

Deployment settings:

OVF file:	C:\Users\Administrator\Desktop\ultra.ova
Download size:	299.4 MB
Size on disk:	8.3 GB
Name:	Cisco CSR 1000V Cloud Services Router
Deployment Configuration:	Small
Host/Cluster:	sj20lab-v109-dhcp3.cisco.com
Datastore:	datastore1 (3)
Disk provisioning:	Thick Provision Lazy Zeroed
Network Mapping:	"GigabitEthernet1" to "VM Network"
Network Mapping:	"GigabitEthernet2" to "VM Network"
Network Mapping:	"GigabitEthernet3" to "VM Network"

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

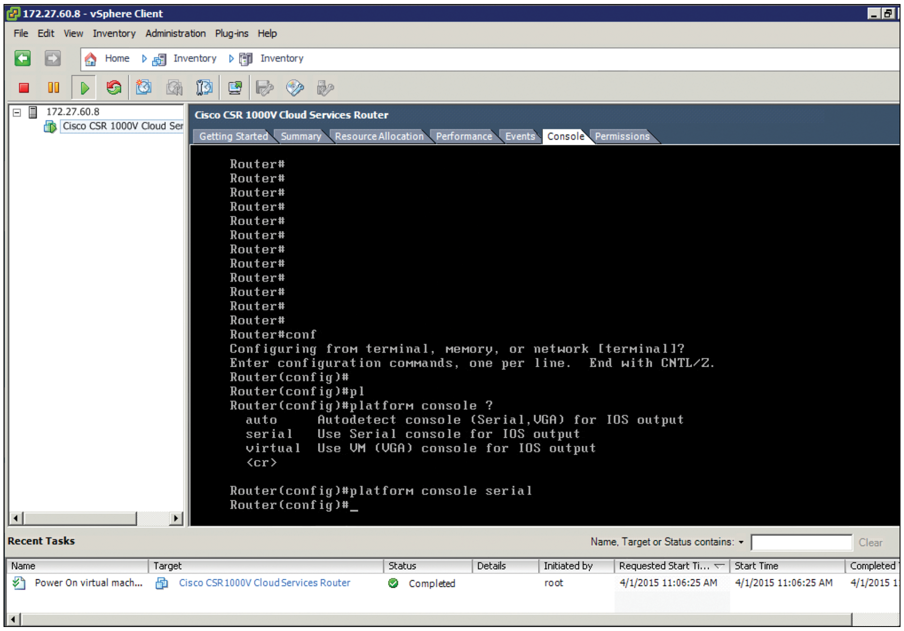
Step 4. When the deployment of the CSR 1000V is complete, boot the router by selecting the VGA console from the GRUB menu on the Console tab shown in Figure 4-16.


172.27.60.8 - vSphere Client
File Edit View Inventory Administration Plug-ins Help
Home Inventory Inventory
172.27.60.8
Cisco CSR 1000V Cloud Ser
Cisco CSR 1000V Cloud Services Router
Getting Started Summary Resource Allocation Performance Events Console Permissions
Recent Tasks
Name, Target or Status contains:
Clear

```
     GNU GRUB  version 0.97  (638K lower / 2619328K upper memory)

 CSR 1000V Autodetect (Serial,VGA) Console -- Mon-25-Aug-14-18:00
 CSR 1000V Virtual (VGA) Console -- Mon-25-Aug-14-18:00
 CSR 1000V Serial Console -- Mon-25-Aug-14-18:00

      Use the ↑ and ↓ keys to select which entry is highlighted.
      Press enter to boot the selected OS, or 'c' for a command-line.

   The highlighted entry will be booted automatically in 7 seconds.
```

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.)


172.27.60.8 - vSphere Client
File Edit View Inventory Administration Plug-ins Help
Home Inventory Inventory
172.27.60.8
Cisco CSR 1000V Cloud Ser
Cisco CSR 1000V Cloud Services Router
Getting Started Summary Resource Allocation Performance Events Console Permissions
Recent Tasks
Name, Target or Status contains:
Clear

```
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)#
Router(config)#pl
Router(config)#platform console ?
  auto      Autodetect console (Serial,VGA) for IOS output
  serial    Use Serial console for IOS output
  virtual   Use VM (VGA) console for IOS output
  <cr>

Router(config)#platform console serial
Router(config)#_
```

Name	Target	Status	Details	Initiated by	Requested Start Ti...	Start Time	Completed
Power On virtual mach...	Cisco CSR 1000V Cloud Services Router	Completed		root	4/1/2015 11:06:25 AM	4/1/2015 11:06:25 AM	4/1/2015 1

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.

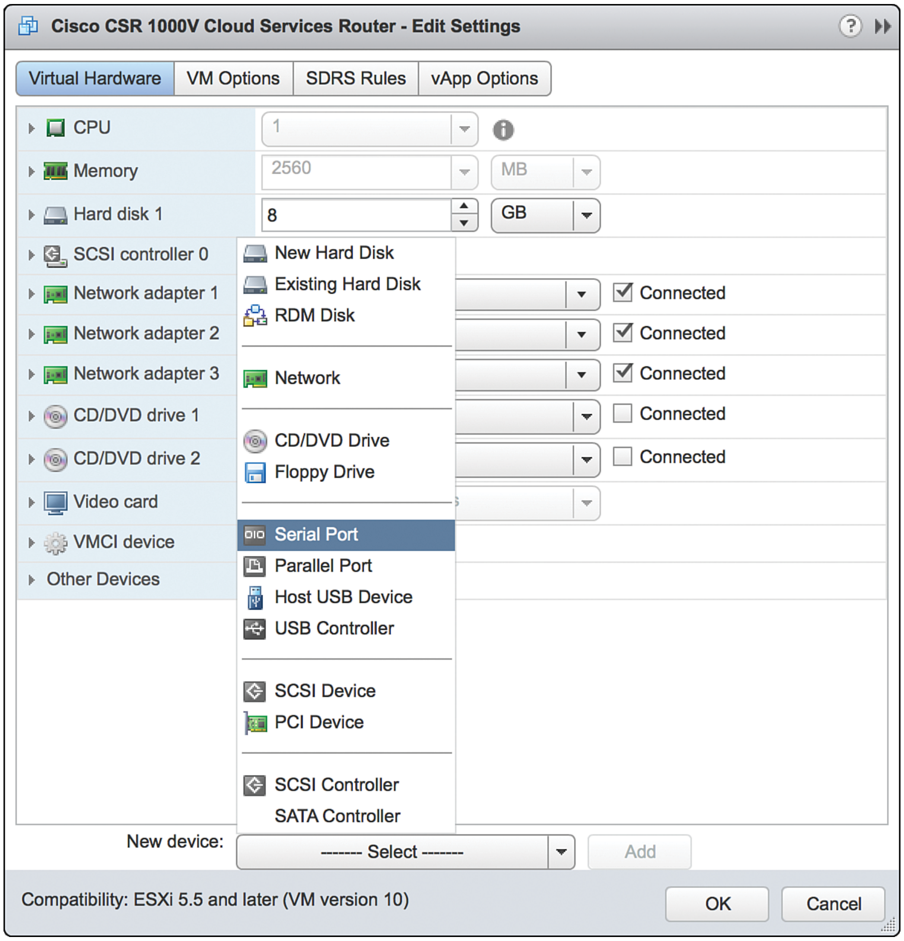
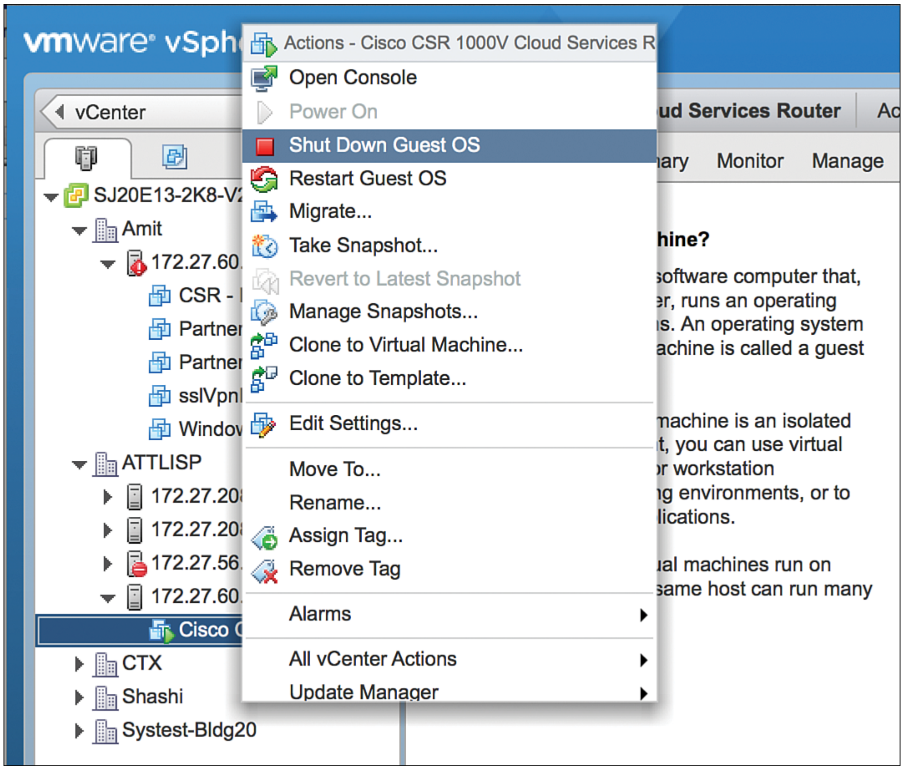


Figure 4-18 *VM Access from the vCenter Web Client*

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.)



hine?

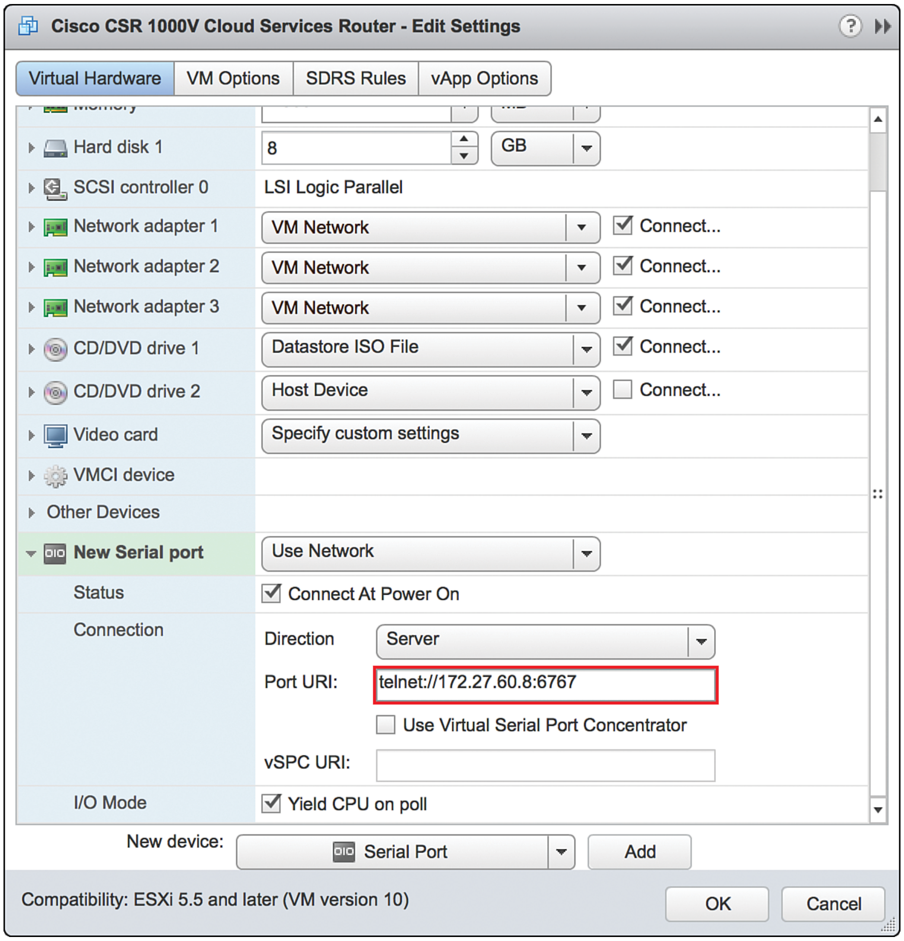
software computer that, er, runs an operating is. An operating system achine is called a guest

machine is an isolated t, you can use virtual r workstation ig environments, or to lications.

ial machines run on same host can run many

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.



Cisco CSR 1000V Cloud Services Router - Edit Settings

Virtual Hardware | VM Options | SDRS Rules | vApp Options

Hard disk 1	8 GB	
SCSI controller 0	LSI Logic Parallel	
Network adapter 1	VM Network	✓ Connect...
Network adapter 2	VM Network	✓ Connect...
Network adapter 3	VM Network	✓ Connect...
CD/DVD drive 1	Datastore ISO File	✓ Connect...
CD/DVD drive 2	Host Device	☐ Connect...
Video card	Specify custom settings	
VMCI device		
Other Devices		
New Serial port	Use Network	
Status	✓ Connect At Power On	
Connection	Direction: Server	
	Port URI: telnet://172.27.60.8:6767	
	☐ Use Virtual Serial Port Concentrator	
	vSPC URI:	
I/O Mode	✓ Yield CPU on poll	

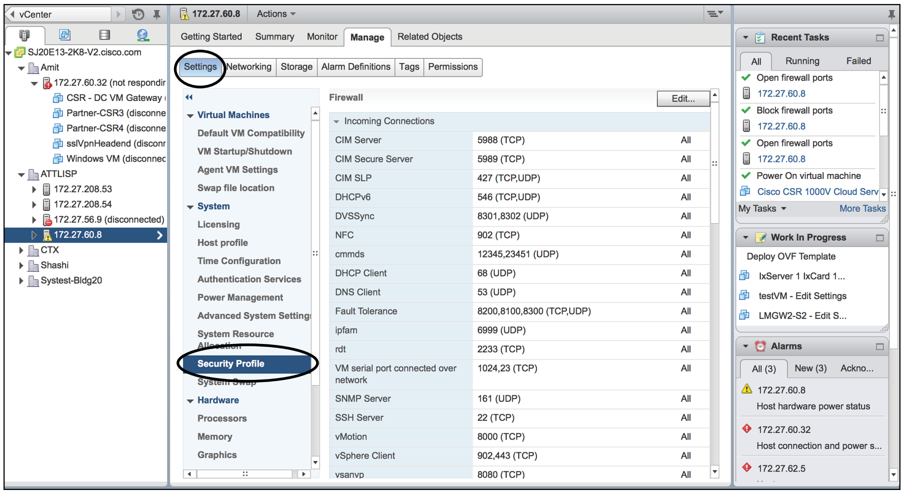
New device: Serial Port | Add

Compatibility: ESXi 5.5 and later (VM version 10)

OK | Cancel

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.



vCenter

SJ20E13-2K8-V2.cisco.com
- Amit
  - 172.27.60.32 (not respondir
    - CSR - DC VM Gateway
    - Partner-CSR3 (disconne
    - Partner-CSR4 (disconne
    - sslVpnHeadend (disconr
    - Windows VM (disconnec
- ATTLISP
  - 172.27.208.53
  - 172.27.208.54
  - 172.27.56.9 (disconnected)
  - 172.27.60.8
- CTX
- Shashi
- Systest-Bldg20

172.27.60.8 Actions

Getting Started | Summary | Monitor | Manage | Related Objects

Settings | Networking | Storage | Alarm Definitions | Tags | Permissions

- Virtual Machines
  - Default VM Compatibility
  - VM Startup/Shutdown
  - Agent VM Settings
  - Swap file location
- System
  - Licensing
  - Host profile
  - Time Configuration
  - Authentication Services
  - Power Management
  - Advanced System Setting
  - System Resource Allocation
  - Security Profile
  - System Swap
- Hardware
  - Processors
  - Memory
  - Graphics

Firewall — Edit...

Incoming Connections

CIM Server	5988 (TCP)	All
CIM Secure Server	5989 (TCP)	All
CIM SLP	427 (TCP,UDP)	All
DHCPv6	546 (TCP,UDP)	All
DVSSync	8301,8302 (UDP)	All
NFC	902 (TCP)	All
cmmds	12345,23451 (UDP)	All
DHCP Client	68 (UDP)	All
DNS Client	53 (UDP)	All
Fault Tolerance	8200,8100,8300 (TCP,UDP)	All
ipfam	6999 (UDP)	All
rdt	2233 (TCP)	All
VM serial port connected over network	1024,23 (TCP)	All
SNMP Server	161 (UDP)	All
SSH Server	22 (TCP)	All
vMotion	8000 (TCP)	All
vSphere Client	902,443 (TCP)	All
vsanvp	8080 (TCP)	All

Recent Tasks — All | Running | Failed

- Open firewall ports — 172.27.60.8
- Block firewall ports — 172.27.60.8
- Open firewall ports — 172.27.60.8
- Power On virtual machine — Cisco CSR 1000V Cloud Serv

My Tasks — More Tasks

Work In Progress

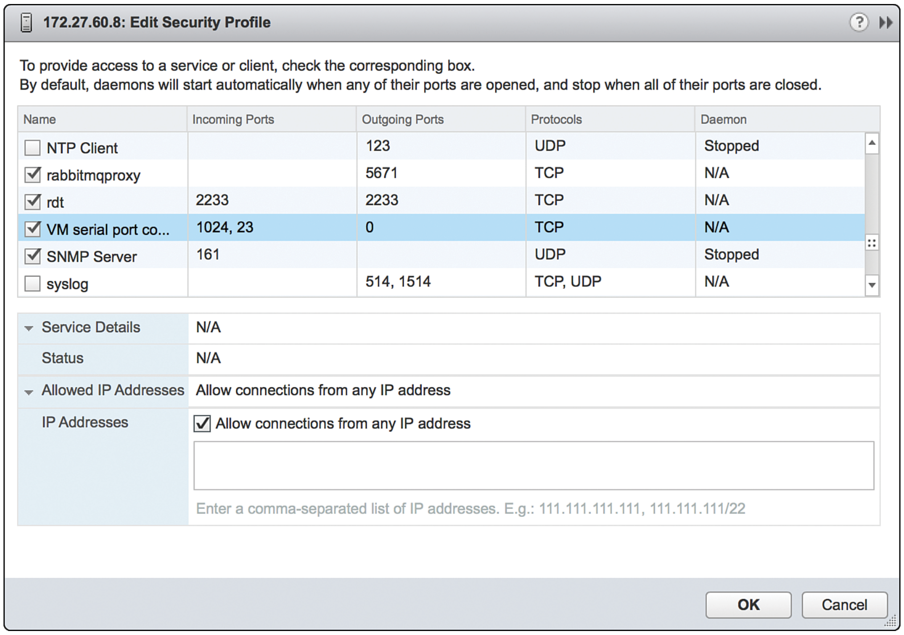
- Deploy OVF Template
- IxServer 1 IxCard 1...
- testVM - Edit Settings
- LMGW2-S2 - Edit S...

Alarms — All (3) | New (3) | Ackno...

- 172.27.60.8 — Host hardware power status
- 172.27.60.32 — Host connection and power s...
- 172.27.62.5

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

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



172.27.60.8: Edit Security Profile

To provide access to a service or client, check the corresponding box.
By default, daemons will start automatically when any of their ports are opened, and stop when all of their ports are closed.

Name	Incoming Ports	Outgoing Ports	Protocols	Daemon
☐ NTP Client		123	UDP	Stopped
☑ rabbitmqproxy		5671	TCP	N/A
☑ rdt	2233	2233	TCP	N/A
☑ VM serial port co...	1024, 23	0	TCP	N/A
☑ SNMP Server	161		UDP	Stopped
☐ syslog		514, 1514	TCP, UDP	N/A

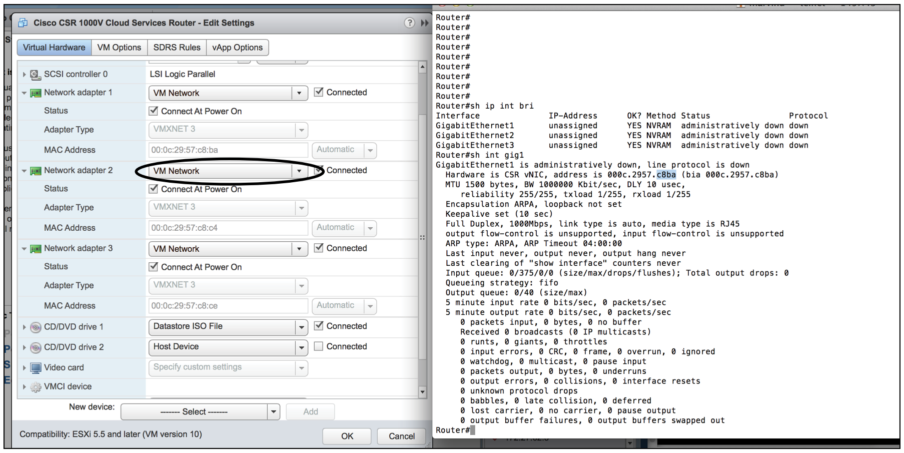
Service Details	N/A
Status	N/A
Allowed IP Addresses	Allow connections from any IP address
IP Addresses	☑ Allow connections from any IP address

Enter a comma-separated list of IP addresses. E.g.: 111.111.111.111, 111.111.111/22

OK | Cancel

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.



Cisco CSR 1000V Cloud Services Router - Edit Settings

SCSI controller 0	LSI Logic Parallel	
Network adapter 1	VM Network	✓ Connected
Status	✓ Connect At Power On	
Adapter Type	VMXNET 3	
MAC Address	00:0c:29:57:c8:ba	Automatic
Network adapter 2	VM Network	✓ Connected
Status	✓ Connect At Power On	
Adapter Type	VMXNET 3	
MAC Address	00:0c:29:57:c8:c4	Automatic
Network adapter 3	VM Network	✓ Connected
Status	✓ Connect At Power On	
Adapter Type	VMXNET 3	
MAC Address	00:0c:29:57:c8:ce	Automatic
CD/DVD drive 1	Datastore ISO File	✓ Connected
CD/DVD drive 2	Host Device	Connected
Video card	Specify custom settings	
VMCI device		

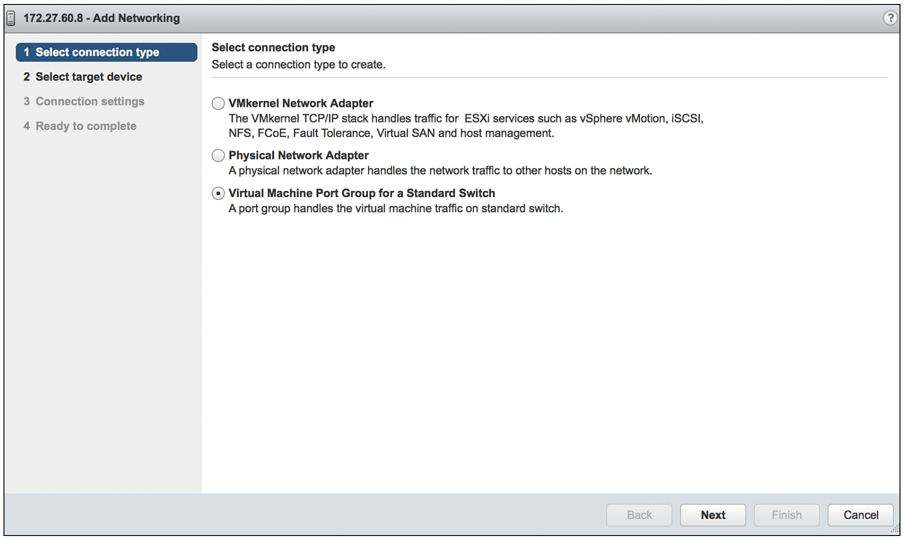
New device: ------- Select ------- Add

Compatibility: ESXi 5.5 and later (VM version 10)

```
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#
Router#sh ip int bri
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       unassigned      YES NVRAM  administratively down down
GigabitEthernet2       unassigned      YES NVRAM  administratively down down
GigabitEthernet3       unassigned      YES NVRAM  administratively down down
Router#sh int gig1
GigabitEthernet1 is administratively down, line protocol is down
  Hardware is CSR vNIC, address is 000c.2957.c8ba (bia 000c.2957.c8ba)
  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 never, output never, 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
     0 packets input, 0 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
     0 packets output, 0 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 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
Router#
```

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.)



172.27.60.8 - Add Networking

Select connection type

Select a connection type to create.

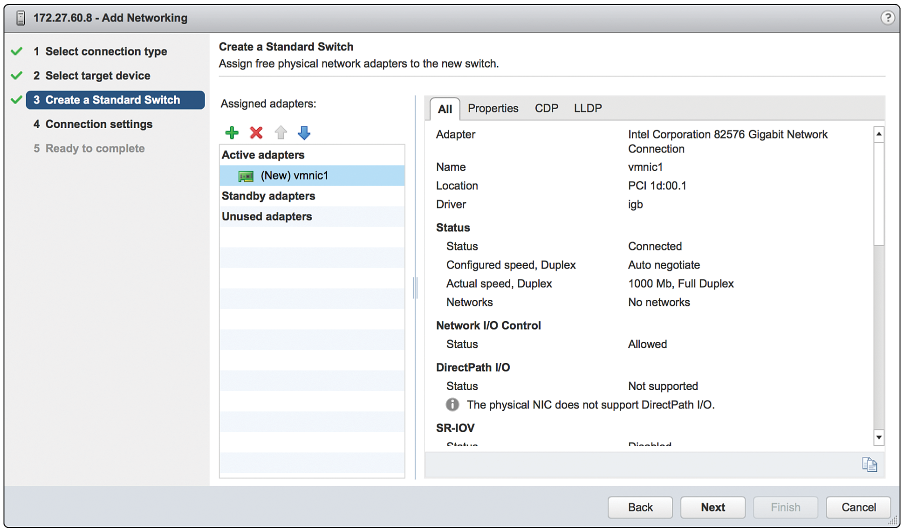
○ **VMkernel Network Adapter**
The VMkernel TCP/IP stack handles traffic for ESXi services such as vSphere vMotion, iSCSI, NFS, FCoE, Fault Tolerance, Virtual SAN and host management.

○ **Physical Network Adapter**
A physical network adapter handles the network traffic to other hosts on the network.

● **Virtual Machine Port Group for a Standard Switch**
A port group handles the virtual machine traffic on standard switch.

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.



Create a Standard Switch
Assign free physical network adapters to the new switch.

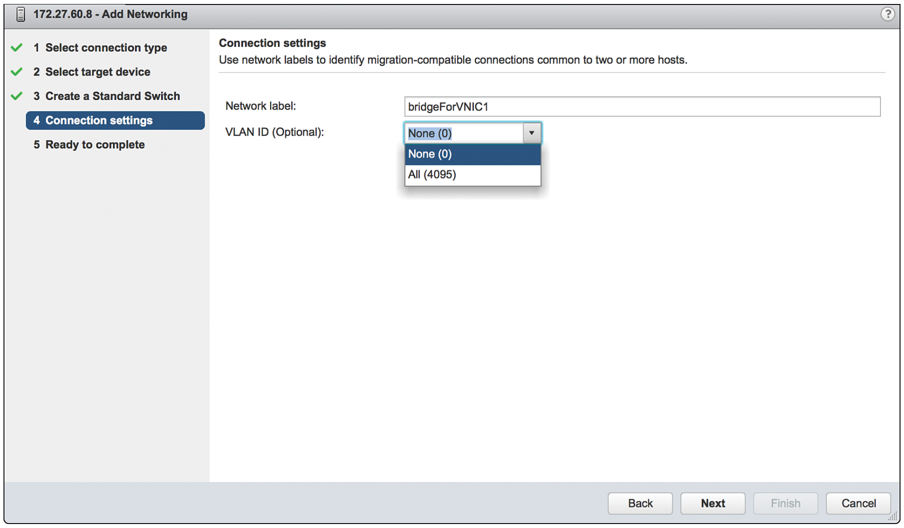
Assigned adapters:

Active adapters
(New) vmnic1
Standby adapters
Unused adapters

Adapter	Intel Corporation 82576 Gigabit Network Connection
Name	vmnic1
Location	PCI 1d:00.1
Driver	igb
Status	
Status	Connected
Configured speed, Duplex	Auto negotiate
Actual speed, Duplex	1000 Mb, Full Duplex
Networks	No networks
Network I/O Control	
Status	Allowed
DirectPath I/O	
Status	Not supported
The physical NIC does not support DirectPath I/O.	
SR-IOV	

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.

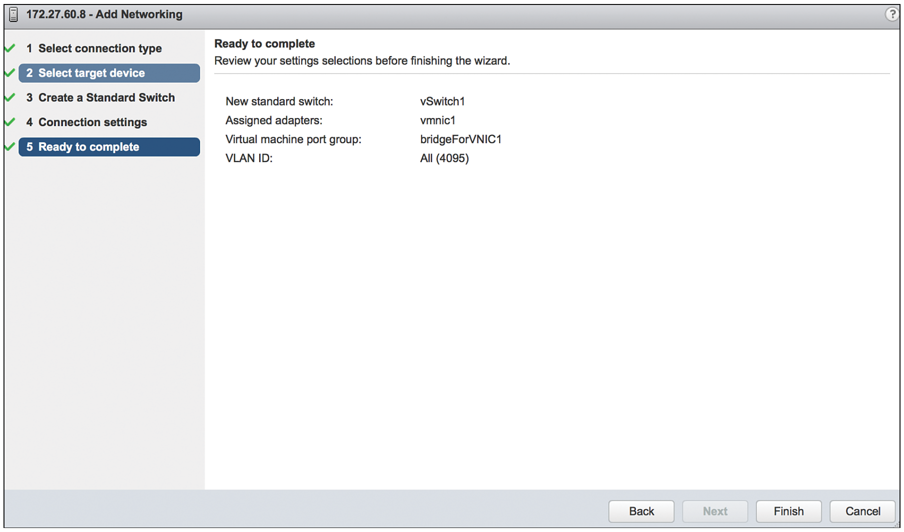


Connection settings
Use network labels to identify migration-compatible connections common to two or more hosts.

Network label: bridgeForVNIC1
VLAN ID (Optional): None (0)
None (0)
All (4095)

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.



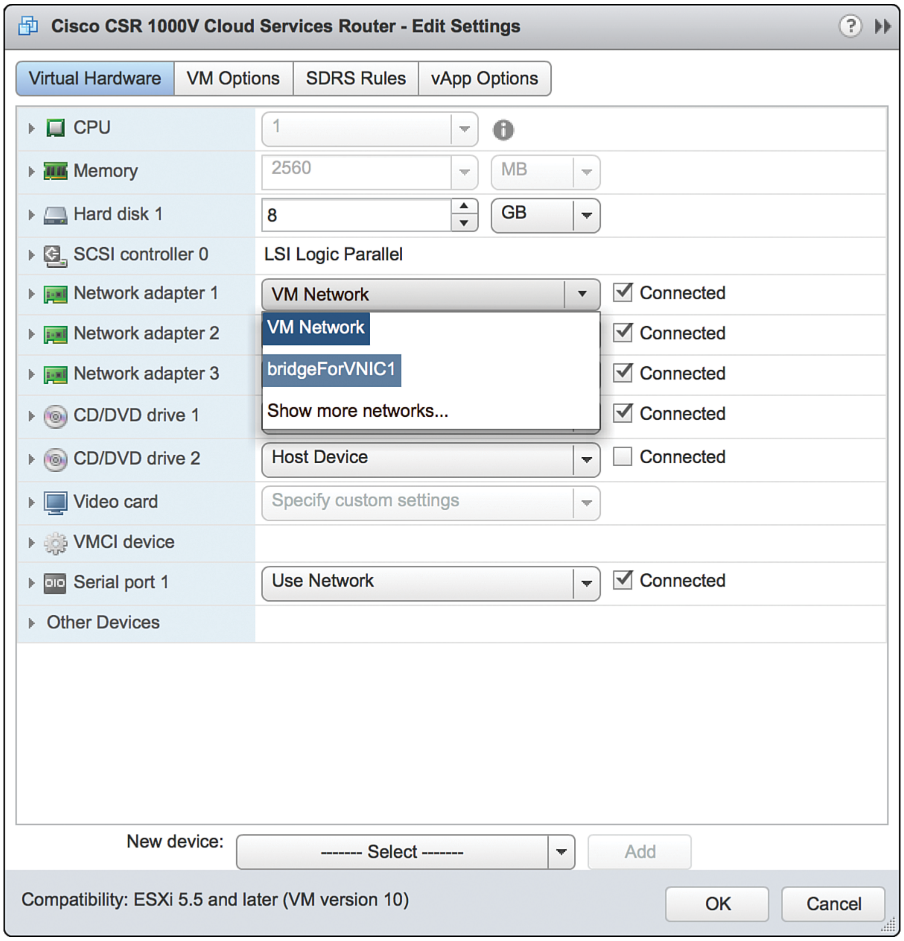
Ready to complete
Review your settings selections before finishing the wizard.

New standard switch:	vSwitch1
Assigned adapters:	vmnic1
Virtual machine port group:	bridgeForVNIC1
VLAN ID:	All (4095)

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.


Cisco CSR 1000V Cloud Services Router - Edit Settings

Virtual Hardware | VM Options | SDRS Rules | vApp Options

CPU	1	
Memory	2560 MB	
Hard disk 1	8 GB	
SCSI controller 0	LSI Logic Parallel	
Network adapter 1	VM Network	✓ Connected
Network adapter 2		✓ Connected
Network adapter 3		✓ Connected
CD/DVD drive 1		✓ Connected
CD/DVD drive 2	Host Device	Connected
Video card	Specify custom settings	
VMCI device		
Serial port 1	Use Network	✓ Connected
Other Devices		

VM Network
bridgeForVNIC1
Show more networks...

New device: ------- Select ------- Add

Compatibility: ESXi 5.5 and later (VM version 10)

OK Cancel

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.

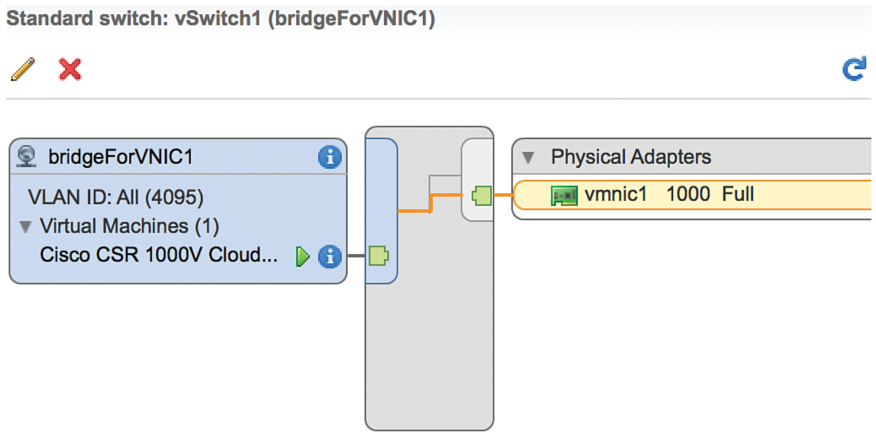


Figure 4-29 *vNICs and the CSR 1000V: Interface Summary Screen*

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:

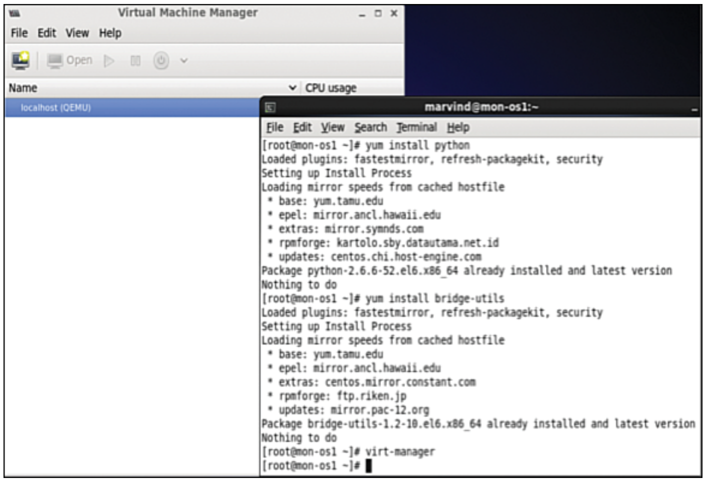
Step 1. Install the VM packages `virt-manager`, `qemu-kvm`, and `bridge-utils` like this:

```
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.



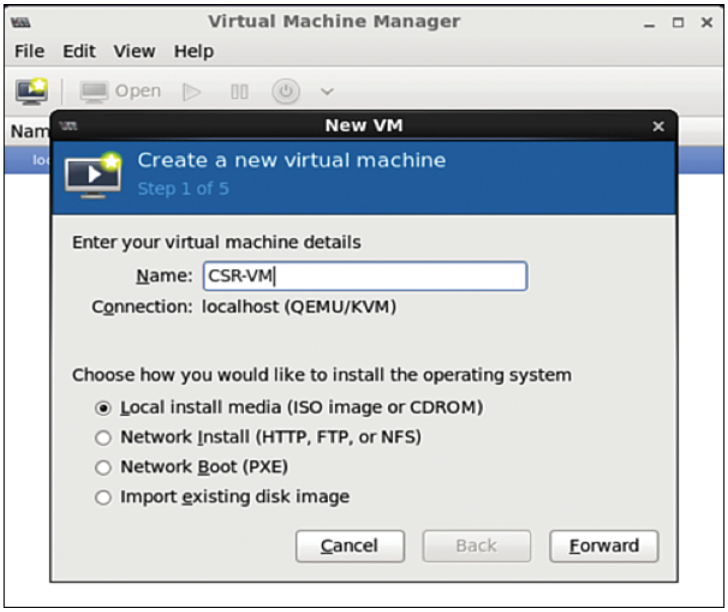
```
[root@mon-os1 ~]# yum install python
Loaded plugins: fastestmirror, refresh-packagekit, security
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: yum.tamu.edu
 * epel: mirror.ancl.hawaii.edu
 * extras: mirror.symnds.com
 * rpmforge: kartolo.sby.datautama.net.id
 * updates: 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.ancl.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.



Enter your virtual machine details

Name: CSR-VM

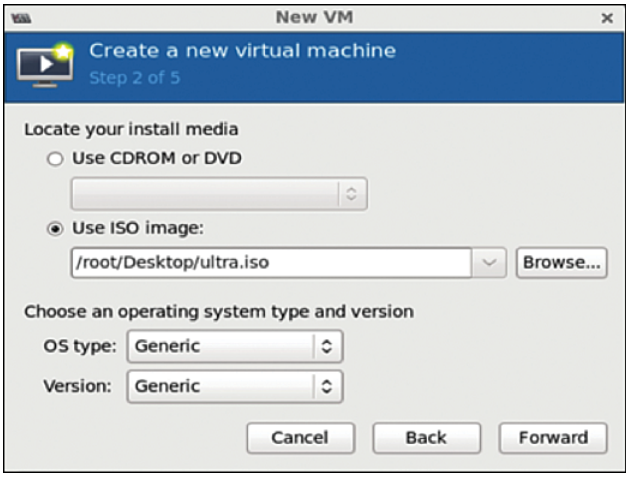
Connection: localhost (QEMU/KVM)

Choose how you would like to install the operating system

- [✓ Local install media (ISO image or CDROM)]
- Network Install (HTTP, FTP, or NFS)
- Network Boot (PXE)
- Import existing disk image

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.



Locate your install media

- Use CDROM or DVD
- [✓ Use ISO image:] /root/Desktop/ultra.iso

Choose an operating system type and version

OS type: Generic

Version: Generic

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.

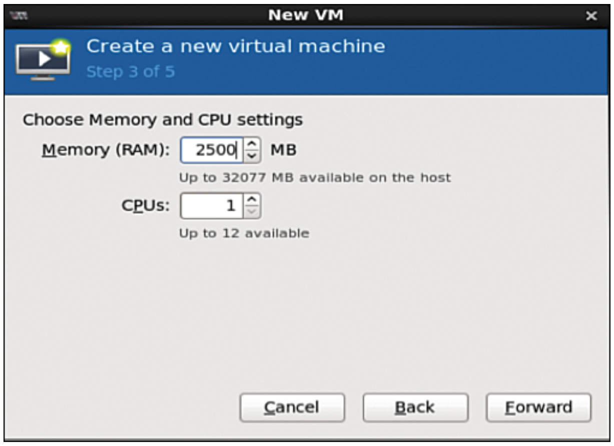

New VM
Create a new virtual machine
Step 3 of 5
Choose Memory and CPU settings
Memory (RAM): 2500 MB
Up to 32077 MB available on the host
CPUs: 1
Up to 12 available

Figure 4-33 *Choosing Memory and CPU Settings*

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

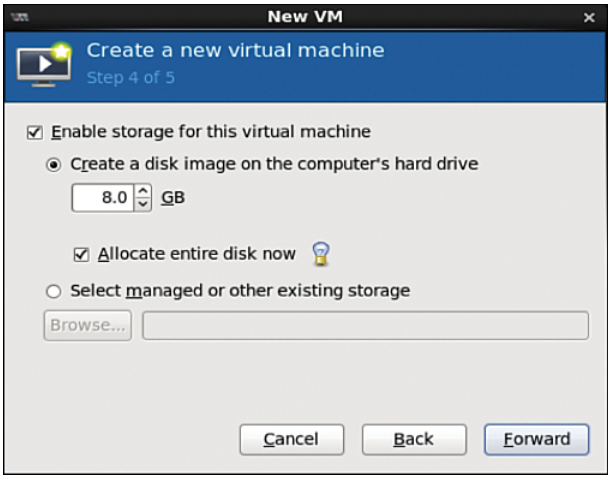
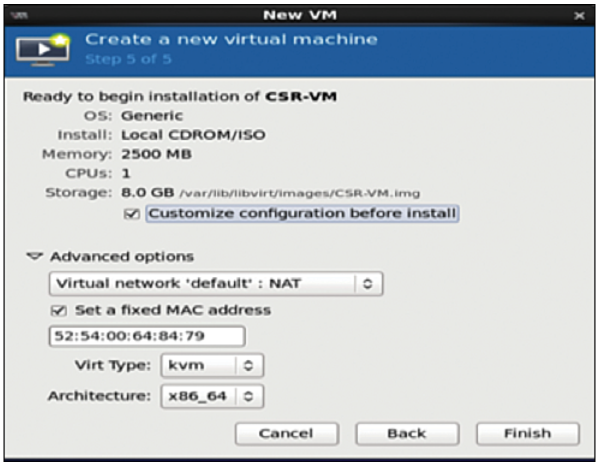

New VM
Create a new virtual machine
Step 4 of 5
☑ Enable storage for this virtual machine
◉ Create a disk image on the computer's hard drive
8.0 GB
☑ Allocate entire disk now
○ Select managed or other existing storage

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.

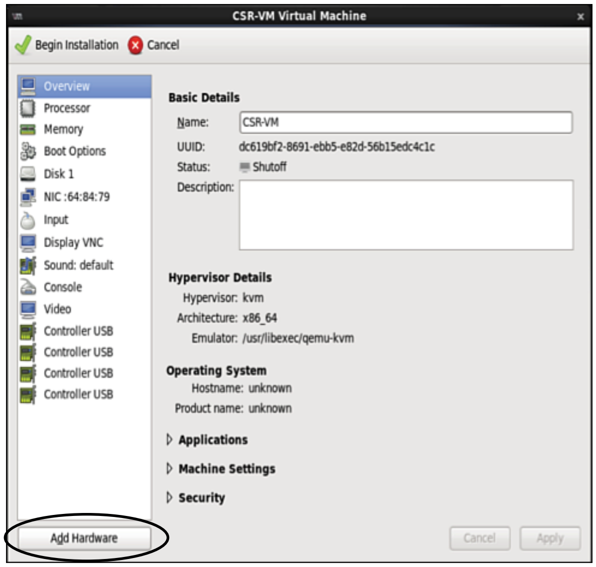


Ready to begin installation of **CSR-VM**
OS: Generic
Install: Local CDROM/ISO
Memory: 2500 MB
CPUs: 1
Storage: 8.0 GB /var/lib/libvirt/images/CSR-VM.img
☑ Customize configuration before install

Advanced options
Virtual network 'default' : NAT
☑ Set a fixed MAC address
52:54:00:64:84:79
Virt Type: kvm
Architecture: x86_64

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.



Basic Details

Name: CSR-VM
UUID: dc619bf2-8691-ebb5-e82d-56b15edc4c1c
Status: Shutoff
Description:

Hypervisor Details

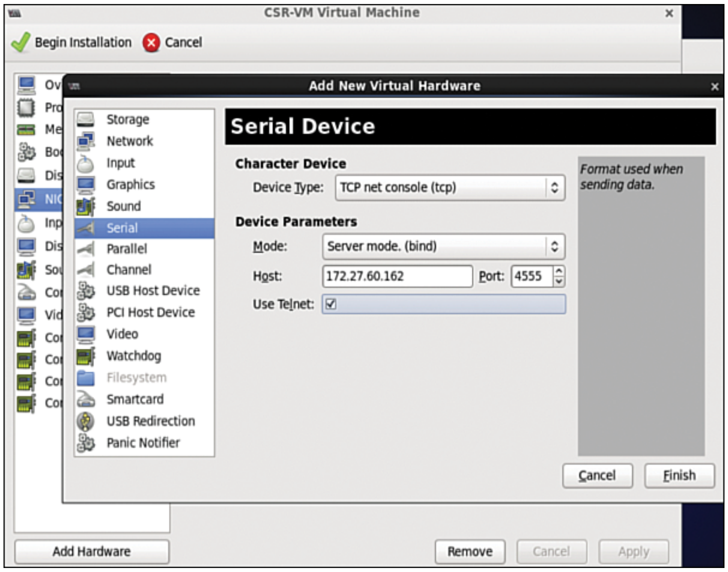
Hypervisor: kvm
Architecture: x86_64
Emulator: /usr/libexec/qemu-kvm

Operating System

Hostname: unknown
Product name: unknown

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.



Serial Device

Character Device

Device Type: TCP net console (tcp)

Device Parameters

Mode: Server mode. (bind)
Host: 172.27.60.162 Port: 4555
Use Telnet: ☑

Format used when sending data.

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.)

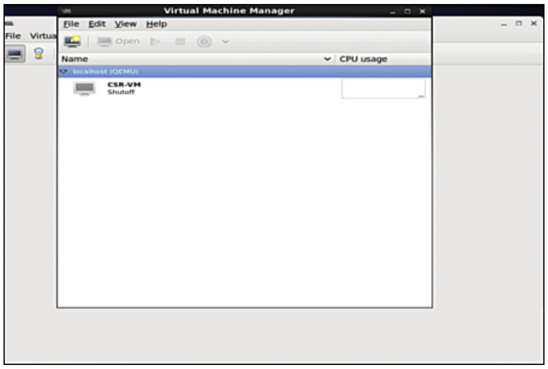


Figure 4-38 *Shutting Down the Guest VM*

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

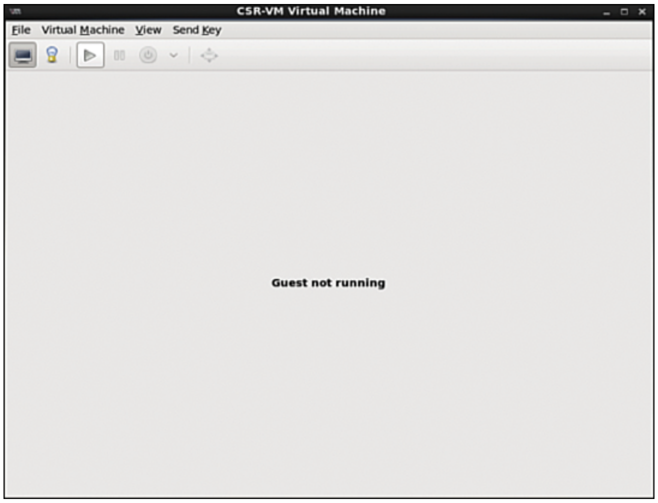
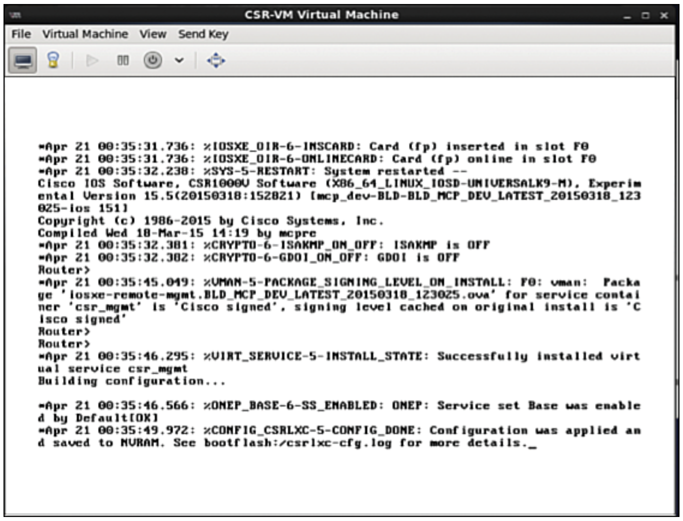


Figure 4-39 *Shutdown of the Guest VM*

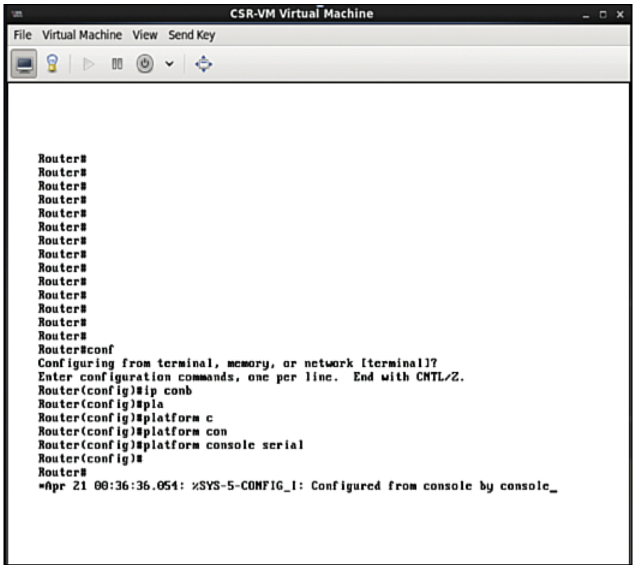
Step 10. Access the router from the console, as shown in Figure 4-40. Make sure the VM is powered up before you try to access it.



```
*Apr 21 00:35:31.736: %IOSXE_OIR-6-INSCARD: Card (fp) inserted in slot F0
*Apr 21 00:35:31.736: %IOSXE_OIR-6-ONLINECARD: Card (fp) online in slot F0
*Apr 21 00:35:32.238: %SYS-5-RESTART: System restarted --
Cisco IOS Software, CSR1000V Software (X86_64_LINUX_IOSD-UNIVERSALK9-M), Experim
ental Version 15.5(20150318:152821) [mcp_dev-BLD-BLD_MCP_DEV_LATEST_20150318_123
025-ios 151]
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: %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', signing level cached on original install is 'C
isco signed'
Router>
Router>
*Apr 21 00:35:46.295: %VIRT_SERVICE-5-INSTALL_STATE: Successfully installed virt
ual service csr_mgmt
Building configuration...

*Apr 21 00:35:46.566: %ONEP_BASE-6-SS_ENABLED: ONEP: Service set Base was enable
d by Default[OK]
*Apr 21 00:35:49.972: %CONFIG_CSRLXC-5-CONFIG_DONE: Configuration was applied an
d saved to NVRAM. See bootflash:/csrlxc-cfg.log for more details._
```

Figure 4-40 *Console Access to the KVM*

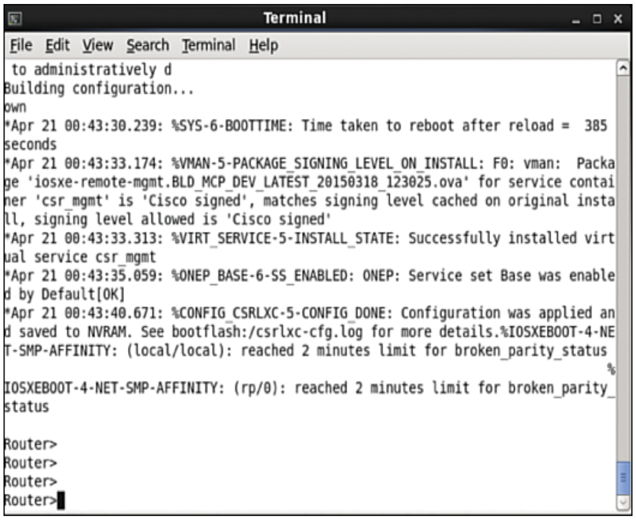
Step 11. Use the serial interface command for telnet access: `platform console serial` and `write mem`, as shown in Figure 4-41.



```
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*

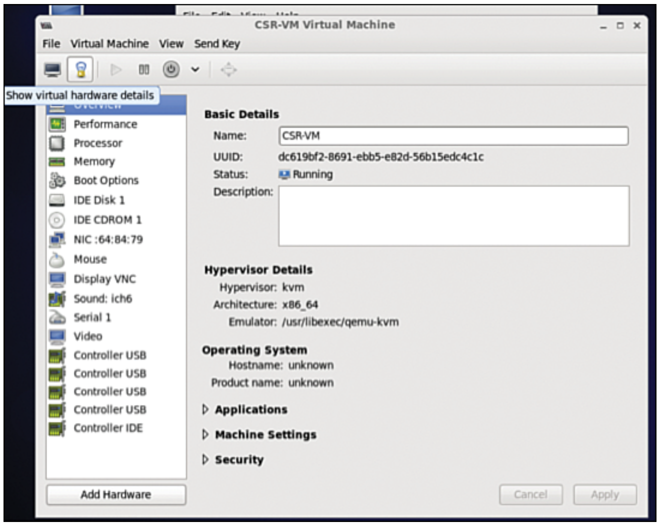
Step 12. Access the CSR 1000V via the telnet, as shown in Figure 4-42.



```
 to administratively d
Building configuration...
own
*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.



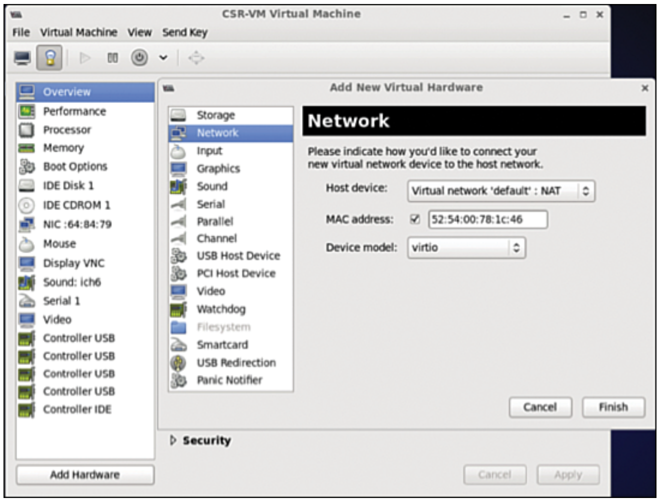
Basic Details
Name: CSR-VM
UUID: dc619bf2-8691-ebb5-e82d-56b15edc4c1c
Status: Running
Description:

Hypervisor Details
Hypervisor: kvm
Architecture: x86_64
Emulator: /usr/libexec/qemu-kvm

Operating System
Hostname: unknown
Product name: unknown

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.



Network

Please indicate how you'd like to connect your new virtual network device to the host network.

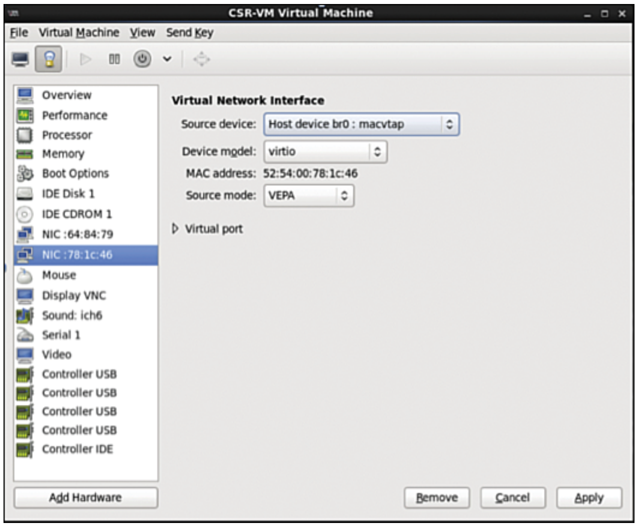
Host device: Virtual network 'default' : NAT

MAC address: ☑ 52:54:00:78:1c:46

Device model: virtio

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.



Virtual Network Interface

Source device: Host device br0 : macvtap

Device model: virtio

MAC address: 52:54:00:78:1c:46

Source mode: VEPA

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).

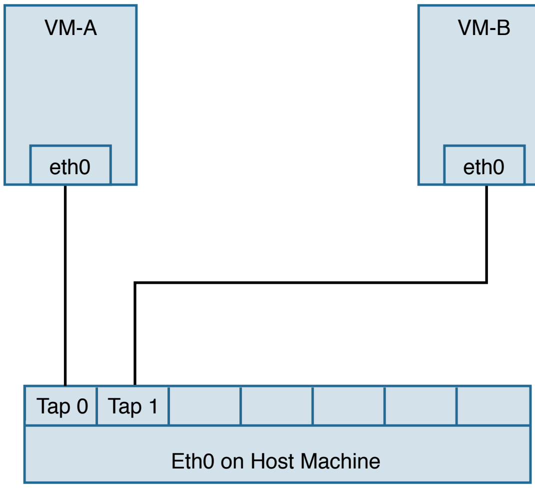
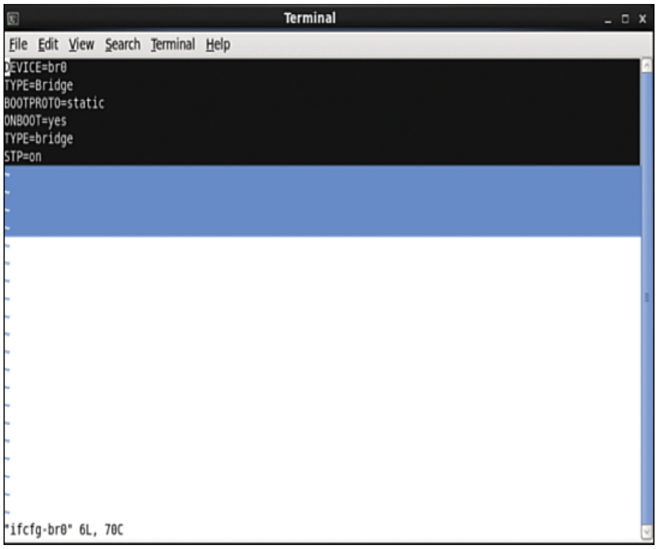


Figure 4-46 `macvtap` *Diagram*

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

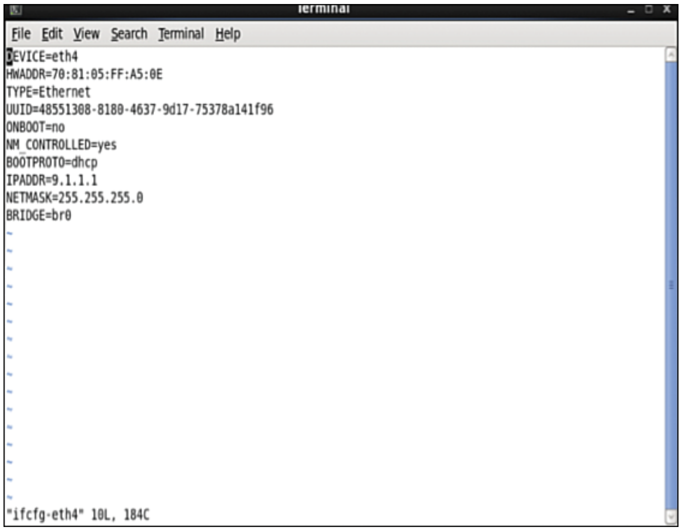
1. Access the directory `/etc/network/interfaces/ifcfg-br0` on the Ubuntu host and view the bridge type (see Figure 4-47).



```
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).



```
DEVICE=eth4
HWADDR=70:81:05: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 pining

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 pining) 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.

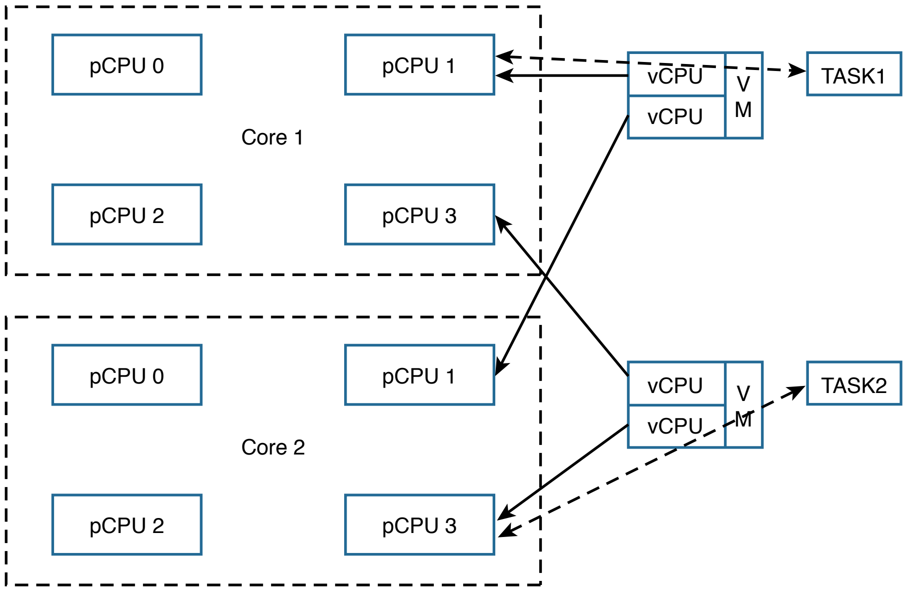


Figure 4-49 *CPU Pining*

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 pining 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 pining 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

Index