Juniper Networks | Engineering Simplicity

Day One Million

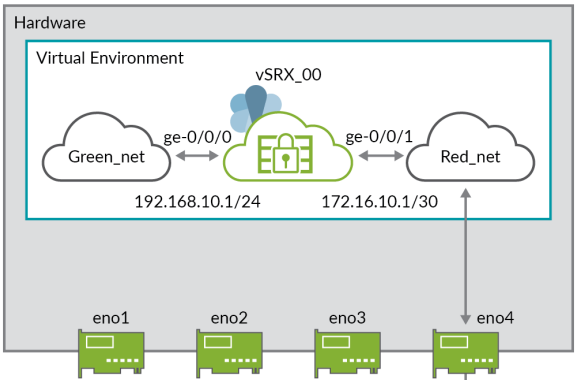
DAY ONE: vSRX on KVM

By Rahul Verma & Madhavi Katti

DAY ONE: vSRX on KVM

Day One: vSRX on KVM is for network administrators, network architects, or engineers interested in quickly starting to use the Juniper Networks vSRX Virtual Firewall. Any time you need to design and test different topology use cases, train yourself or others, or even practice certification exams, this book covers such usage with step-by-step instructions and practical examples.

Day One: vSRX on KVM requires basic networking knowledge and a general understanding of the TCP/IP protocol suite, Linux systems, and Ubuntu. Written in tandem with the Juniper vSRX documentation, it curates links and tutorials with the Juniper TechLibrary and saves time for vSRX users by coordinating deployment steps with the TechLibrary's archives. Learn how to deploy vSRX instances today!



IT'S DAY ONE AND YOU HAVE A JOB TO DO, SO LEARN HOW TO:

- Install vSRX's prerequisite packages and configure and deploy an instance of vSRX on KVM.
- Create a single instance topology and then a multi-device topology using two vSRX instances.
- Design topologies for different use cases.
- Complete the three challenge topologies.
- Troubleshoot vSRX operations.

ISBN 978-1941441893

Juniper Networks Books are focused on network reliability and efficiency. Peruse the complete library at www.juniper.net/books.

JUNIPER NETWORKS

Day One: vSRX on KVM

by Rahul Verma and Madhavi Katti

Day One Million

JUNIPER NETWORKS

Published by Juniper Networks Books
Authors: Rahul Verma, Madhavi Katti
Technical Reviewers: Casper Rijnders, Vikas Singh, Jayadevi Santhanagopalan, Pramod Nellikka, Vikas Vishwanathan, Antoine Taza
Editor in Chief: Patrick Ames
Copyeditor: Nancy Koerbel
Illustrator: Karen Joice
Project Management: Indira Upadhayaya

ISBN: 978-1-941441-89-3 (print)
Printed in the USA by Vervante Corporation.

ISBN: 978-1-941441-88-6 (ebook)

Version History: v1, April 2019
2 3 4 5 6 7 8 9 10

http://www.juniper.net/dayone

About the Authors
Rahul Verma is a CFTS engineer based in Bengaluru, India. He has 10 years of experience working with different Juniper product lines, mainly ScreenOS and Junos (SRX and vSRX). This is his first Day One, but in his many years of work as a Technical Support Engineer, he's seen how important the Day One series is for newbies.

Madhavi Katti is an Information Development Engineer at Juniper Networks with over 10 years of experience in writing and developing documentation for networking and telecommunications. Madhavi contributes to product documentation for security and virtualization products.

Authors' Acknowledgments
We would like to thank Patrick Ames and Nancy Koerbel for guidance on writing for the Day One series. We would also like to thank the technical reviewers and JTAC for looking over our words and offering plenty of encouragement along the way. Thanks to Karen Joice for support in developing illustrations, and special thanks to our managers Indira Upadhayaya, Sujit Nair, and Aditya Maheshwari for their vision, support, and encouragement.

Feedback? Comments? Error reports? Email them to dayone@juniper.net.

Welcome to Day One

This book is part of the *Day One* library, produced and published by Juniper Networks Books.

Day One books cover the Junos OS and Juniper Networks networking essentials with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow. You can obtain the books from various sources:

- Download a free PDF edition at http://www.juniper.net/dayone.
- Many of the library's books are available on the Juniper app: Junos Genius.
- Get the ebook edition for iPhones and iPads from the iBooks Store. Search for *Juniper Networks Books* or the title of this book.
- Get the ebook edition for any device that runs the Kindle app (Android, Kindle, iPad, PC, or Mac) by opening your device's Kindle app and going to the Amazon Kindle Store. Search for *Juniper Networks Books* or the title of this book.
- Purchase the paper edition at Vervante Corporation (www.vervante.com) for between $15-$40, depending on page length.
- Note that most mobile devices can also view PDF files.

TechLibrary Connection

This *Day One* book makes a direct connection to the *Juniper TechLibrary* and all of its security docs for both the SRX Series and vSRX. Here are some vital starting points to visit in the TechLibrary (and throughout this book you'll find dozens more curated links that point to other instructional content you might consider):

Security Products and Solutions: https://www.juniper.net/us/en/products-services/security/.

SRX Series Chassis Cluster Configuration Overview: https://www.juniper.net/documentation/en_US/junos/topics/task/operational/chassis-cluster-srx-series-creating.html.

Check the latest vSRX specs: https://www.juniper.net/us/en/products-services/security/srx-series/vsrx/.

All vSRX documentation starts here: https://www.juniper.net/documentation/product/en_US/vsrx.

Download vSRX here: https://www.juniper.net/us/en/dm/free-vsrx-trial/.

What You Need to Know Before Reading This Book

You should be familiar with the basic administrative functions of Junos OS and UNIX, including the ability to work with operational commands and to read, understand, and change configurations.

There are several books in the *Day One* library on learning Junos, at http://www.juniper.net/dayone.

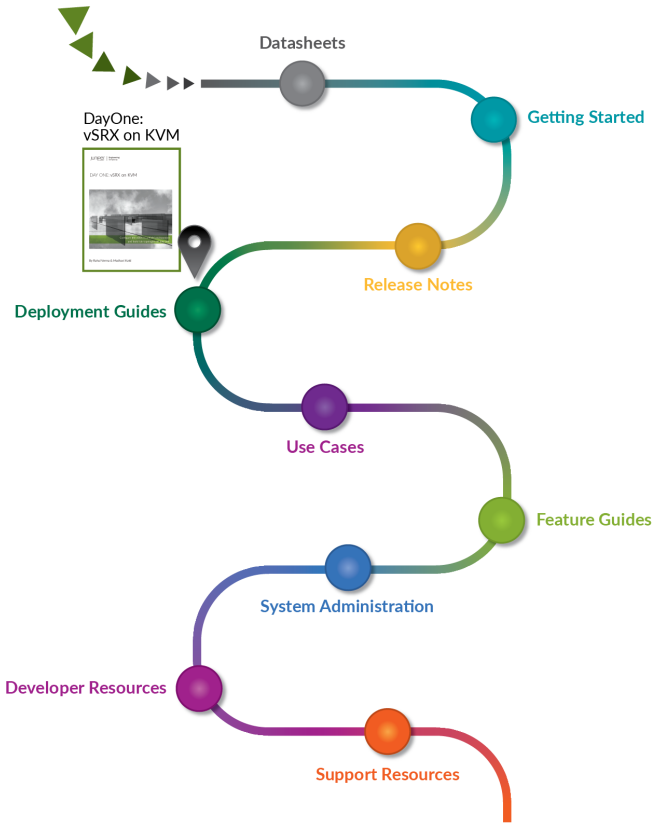
This book assumes that you, the reader, have intermediate-level knowledge of:

- Basic networking and an understanding of the TCP/IP protocol.
- Linux system administration (preferably Ubuntu), and knowledge of the Linux virtualization solution (KVM).
- Junos OS operational and configuration modes.
- Junos OS and how to use its CLI.
- Configuration of feature sets on SRX Series devices.

What You Will Learn by Reading This Book

- Understand the architecture of the vSRX, its specifications, and its licensing models.
- Install vSRX's prerequisite packages, and configure and deploy an instance of vSRX on KVM.
- Create a single instance topology and then a multi-device topology using two vSRX instances.
- Design topologies for different use cases.
- Complete the three challenge topologies.
- Troubleshoot vSRX operations.

vSRX
DOCUMENTATION PATH



All Things vSRX

- vSRX virtual firewall product page: https://www.juniper.net/us/en/products-services/security/srx-series/vsrx/
- List of supported features on vSRX in Junos OS Release, Feature Explorer: https://apps.juniper.net/feature-explorer/select-platform.html?category=Security&typ=1#pid=20600616&platform=vSRX
- vSRX product datasheet: https://www.juniper.net/assets/us/en/local/pdf/datasheets/1000489-en.pdf
- vSRX in the AWS Marketplace: https://aws.amazon.com/marketplace/pp/B01LYWCGDX/
- Try vSRX in vLabs: https://jlabs.juniper.net/vlabs

More vSRX product documentation:

- AWS: https://www.juniper.net/documentation/en_US/vsrx/information-products/pathway-pages/security-vsrx-aws-guide-pwp.html
- KVM: https://www.juniper.net/documentation/en_US/vsrx/information-products/pathway-pages/security-vsrx-kvm-guide-pwp.html
- Microsoft Azure: https://www.juniper.net/documentation/en_US/vsrx/information-products/pathway-pages/security-vsrx-azure-guide-pwp.html
- Contrail: https://www.juniper.net/documentation/en_US/vsrx/information-products/pathway-pages/security-vsrx-contrail-guide-pwp.html
- VMWare: https://www.juniper.net/documentation/en_US/vsrx/information-products/pathway-pages/security-vsrx-vmware-guide-pwp.html
- Microsoft Hyper-V: https://www.juniper.net/documentation/en_US/vsrx/information-products/pathway-pages/security-vsrx-hyper-v-guide-pwp.html

Chapter 1

Introduction to vSRX on KVM

This chapter reviews virtualization in a nutshell and compares a traditional physical architecture with a virtual one. It then compares a physical network with a virtual network, discussing the components involved and what changes in the transition from physical to virtual. This follows with a virtual form of the SRX Series, the vSRX, its basic components, and how it communicates. The chapter concludes by detailing the minimum hardware and software requirements for installing vSRX on KVM and a brief on the licensing model.

Virtualization fundamentally centralizes administrative tasks while improving scalability and workloads, which can lead to the consolidation of network infrastructure, lower cost, greater security, ease of management, and other benefits.

Consider a scenario where there are no public transportation systems, such as railways and buses, and millions of people driving their own vehicles to reach their destination. What happens? Frequent traffic congestion, increased use of fuels, more air pollution, and a waste of everyone's time.

Public transportation systems save lots of resources compared to every passenger driving their own vehicle.

If you compare virtualization to a public transport system, then the physical host is a train or bus and the virtual machines are the passengers. Adopting to virtualization means that instead of using multiple computers running on their own hardware (everyone has their own car), everything is moved to a single server that acts as a host and runs virtual instances of multiple computers.

Let's start with understanding how virtualization works and how networks are virtualized.

Virtualization in a Nutshell

Virtualization can be defined as the creation of multiple virtual resources from one physical resource. This is similar to one physical system performing the same function as that of multiple physical systems.

Consider this example: you and your colleague share a project built on Windows and have an executable in ".exe" format. *What do you do? How do you run it? Will you go to IT and say I need a Windows machine to run that executable?*

What if you can run Windows in a virtualized environment? It could be the solution you're looking for, and a fruitful one, too, saving you time and resources.

Yes, this is doable. All you need is a special software package that allows virtualized environments to be built on top of your host machine.

Your laptop is hardware (a metal or plastic box that you carry in your backpack) and the moment you power it up, it boots up with installed software (let's say MacOS). This software gives you the look and feel of the host machine. Now, to enable a virtualized environment, the special software you need is known as a *hypervisor*. A hypervisor sits between the hardware and software layer and allows for the host to be virtualized. This is what allows Windows to run on MacOS.

Figure 1.1 further illustrates how a hypervisor alters a traditional architecture.

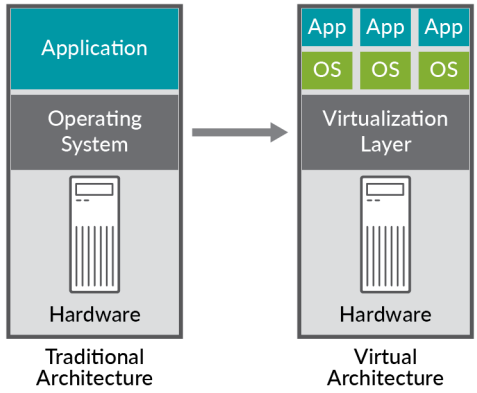


Figure 1.1 *Traditional and Virtual Architecture*

On the left side of Figure 1.1 you have a traditional architecture composed of the underlying host hardware, the host operating system (OS) installed, and applications running on the OS. When you compare this to virtualized architecture, you have a virtualization layer that fits between the hardware and the OS.

The host (let's say an x86 hardware) contains all the physical interface cards, CPUs, and memory, and also contains the base operating system (for example, Ubuntu). On top of this you deploy the hypervisor, and then install Windows in a virtual form—or let's say SRX—in a virtual form factor.

It's the hypervisor that exposes the underlining hardware resources and partitions your physical server hardware into multiple virtual machines (VM). VMs are an instance created by utilizing the physical hardware resources.

Multiple VMs can run on top of a host machine and share the same physical host resources, and they act like a real computer with their operating system and devices (virtual hardware – CPUs, Memory, I/O).

Comparing a Physical Network with a Virtual Network

Let's discuss what changes from the physical networking prospective when you add virtual resources. Figure 1.2 visualizes a simple enterprise network – a typical office network.

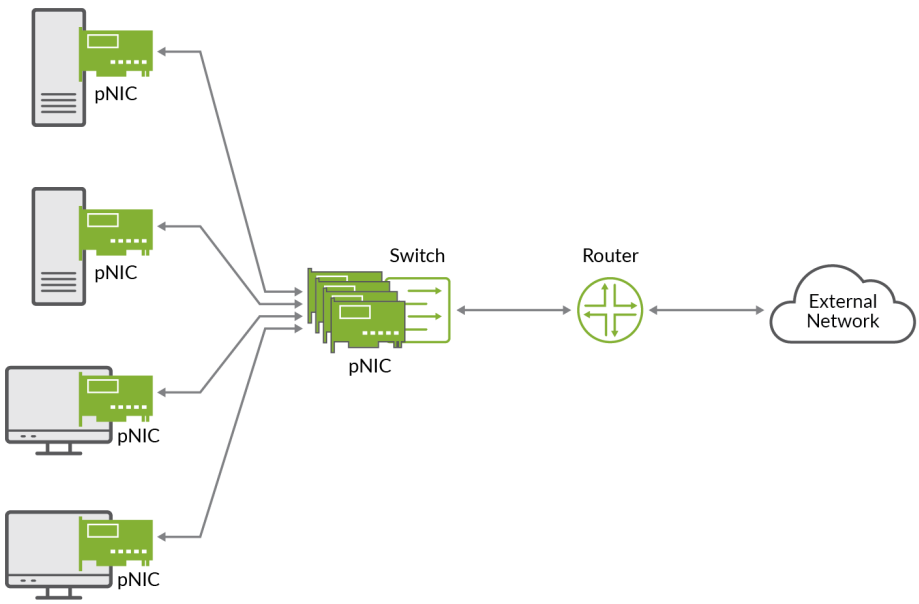


Figure 1.2 *Physical Network Topology*

The components of Figure 1.2 are:

- End user devices: laptops, desktops, guest user devices.
- Physical NIC (pNIC): network interface cards on end user devices.
- Servers: database server, ticketing tool server, authentication server.
- Layer 2 switches: connecting user machines to servers.
- External network: the switch is usually connected to a cable or DSL modem or router which provides Internet access to end user devices.

Physical servers use one or multiple network interfaces cards (NICs). Those physical NICs connect to physical switch ports. NIC communicates with other NICs in the same network using a network switch, and also when connecting to a different network, such as the Internet. The switch is connected to a router that allows networks to communicate with each other.

However, in a virtual world most of the physical components get converted into a virtual component. Figure 1.3 captures what those changes are.

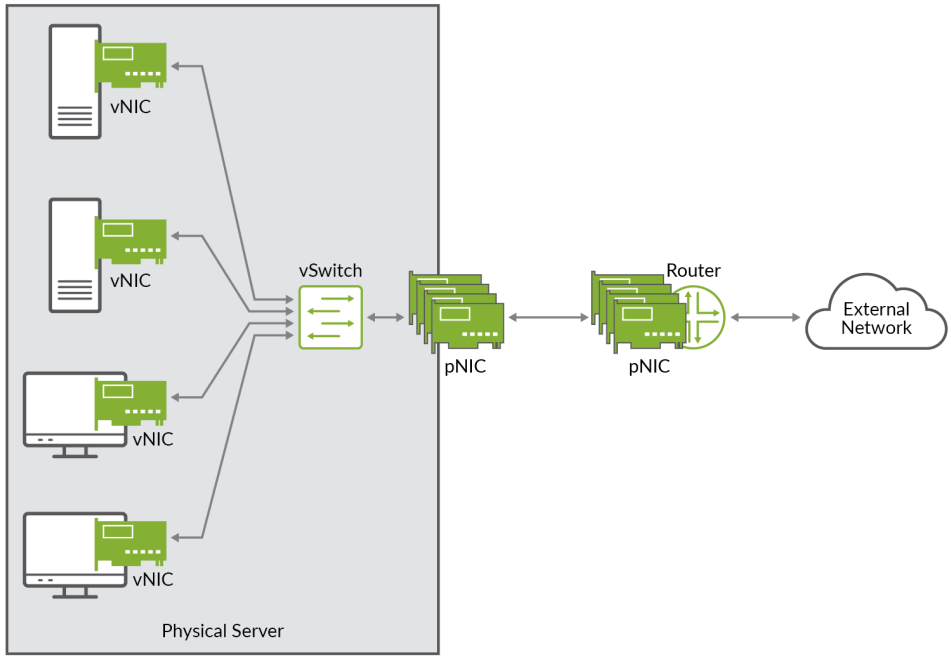


Figure 1.3 *Network with Open vSwitch or Linux Bridge*

And the components of Figure 1.3 are:

- Host server: physical server runs the operating systems and hypervisor software.
- Virtual end user devices: in virtual networks, these are VMs run as a software entity within the host server.
- Virtual NIC: VMs have vNICs connecting them to a virtual switch.
- Virtual switch: virtual switches provide inter-VM connectivity as well as external access to a physical switch.
- Physical NIC: the physical NIC are installed on physical host servers and support network connectivity to external networks.
- External network: the switch is usually connected to a cable or DSL modem or router which provides Internet access to end user devices.

You can see the difference that virtualization brings to the plate as compared to a physical networking setup.

In virtual networks, virtual devices and VMs are connected to virtual switches through vNICs.

How does a virtual switch provide external physical network access or Internet connectivity to virtual machines?

The answer is that the virtual switch uses the pNICs associated with the host server to connect the virtual network to the physical network.

Network functions like routing, switching, firewalls, load balancing, and many more are being virtualized because of the cost savings that virtualization brings to the table. The Juniper virtualized platform for security is vSRX, with other functions represented by the vMX in the routing sector and the vQFX in the switching sector.

Introduction to vSRX

The vSRX is a virtual Juniper Networks SRX Series firewall that is optimized to run as software on x86 servers. Like other physical SRX Series devices, the vSRX runs on Junos OS and offers the same features as the SRX Series firewalls.

The vSRX can be installed on any server hardware of your choice, as long as it is x86-based with an Intel Nehalem or newer generation CPU, and running KVM or VMware.

vSRX Architecture

Let's briefly review the basic architecture of the vSRX before installing and configuring it. Figure 1.4 shows the building blocks of the vSRX virtual firewall.

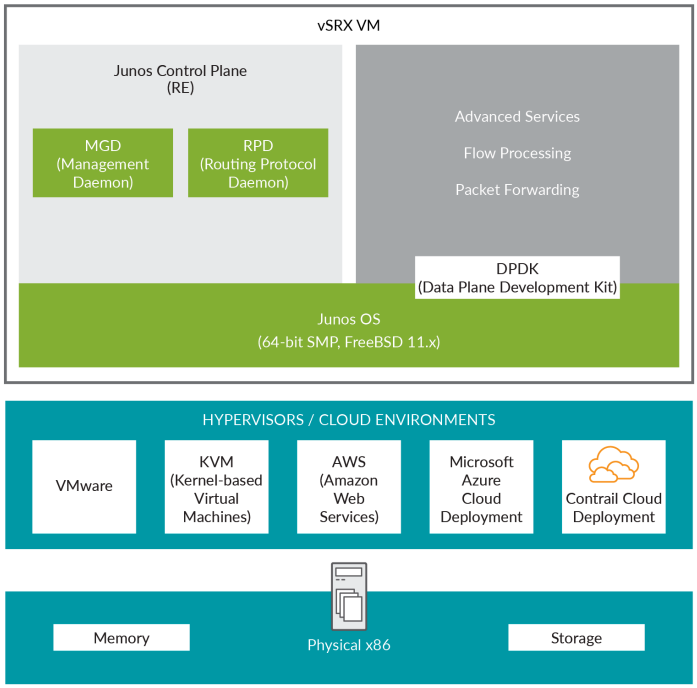


Figure 1-4 vSRX Architecture

NOTE This architecture diagram and this book are based on the vSRX3.0, which is supported from Junos OS Release 18.4R1 onwards.

MORE? Always check the vSRX product pages for the latest iterations of vSRX releases and the TechLibrary's Release Notes: https://www.juniper.net/documentation/product/en_US/vsrx and https://www.juniper.net/documentation/en_US/vsrx/information-products/topic-collections/release-notes/18.4/index.html.

Table 1.1 provides details on the components of the vSRX architecture.

Table 1.1 vSRX Architecture Components

Component	Description
Physical X86	The server at the hardware layer contains the physical network interface cards (NICs), CPUs, and memory. This can be any industry standard x86 servers (running Intel processors) that support virtualization capabilities.
Hypervisors	Over the hardware layer, kernel-based virtual machine (KVM), VMware ESXi provides the host environment for vSRX to run as a VM. This manages the boot complex, CPU memory storage, and various other hardware components of the host.
Guest OS	Junos OS runs as a guest OS; it runs the control plane as Routing Engine and the data plane as Packet Forwarding Engine. The Packet Forwarding Engine does utilize DPDK for higher performance.
vCPU	Represents the logical CPU virtualized by the Intel x86 64-bit CPU. vSRX uses one virtual CPU (vCPU) for the Routing Engine and at least one vCPU for the Packet Forwarding Engine.
Management process MGD/ Routing protocol process (RPD)	MGD provides communication between the other processes and an interface to the configuration database. RPD defines how routing protocols such as RIP, OSPF, and BGP operate on the device, including selecting routes and maintaining forwarding tables.
Packet Forwarding	Processes packets and applies filters, routing policies, and other security features.
DPDK	A set of data plane libraries and network interface controller drivers for fast packet processing on Intel IA platform. Supports para-virtualized NIC drivers like Virtio VMXNET3, and direct I/O like SR-IOV.

You can install the vSRX virtual firewall on:

- KVM
- VMware ESXi,
- Juniper Networks Contrail
- Amazon Web Services (AWS) cloud
- Microsoft Azure Cloud

MORE? For a list of up-to-date supported platforms for vSRX, see Juniper's Feature Explorer application:

- https://apps.juniper.net/home/#vSRX/Features
- https://apps.juniper.net/feature-explorer/parent-feature-info.html?pFName=Virtualization.

How vSRX Communicates

When you talk about interfaces, something like 1-Gigabit Ethernet (ge) or 10-Gigabit Ethernet (xe) interface, comes to mind. Since this is a virtualized environment, it's wise to also learn about the virtualized terminologies about virtualized interfaces.

This *Day One* book focuses on KVM running on top of Ubuntu as its host OS, so let's understand the interfaces within this context and you'll know the terms we're using.

A VM NIC (also known as a vNIC) on the KVM hypervisor, is known as a VIRTIO interface and uses the keyword "vnet" to define it in configuration.

When two VMs are running on the same host and you want them to communicate, this communication is provided by one of the following mediums:

- Linux bridge
- Open Virtual Switch (OVS) – (Out of scope for this *Day One* book)
- SR-IOV (Single Root IO Virtualization)
- PCI pass-through

Figure 1.5 illustrates these virtual interfaces.

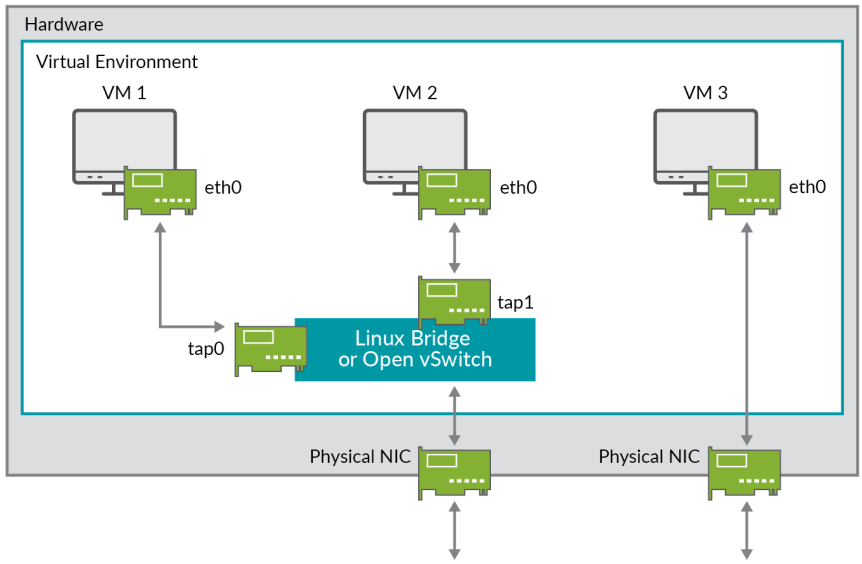


Figure1.5 Network with OpenVswitch or Linux Bridge

The virtual switch (Linux Bridge) works on lines similar to that of a physical switch and assists in communication between multiple VMs that are connected to it. The virtual switch can also have a connection to a physical NIC if the traffic is required to flow outside the network. In a high-performance scenario, that is, at throughput requirements of 3Gbps or greater for a VM, VIRTIO connectivity is not feasible. vSRX supports pass-through of the virtual switch by directly communicating with the physical NIC.

There are two supported variants:

- SR-IOV – This variant allows a physical function to appear as multiple vNICs, appearing as virtual functions.
- PCI-pass-through – This variant allows a physical function to appear directly for a VM, bypassing the KVM hypervisor completely.

MORE? You can find information about OVS at: http://www.openvswitch.org/.

vSRX Minimum Hardware and Software Requirements

Before you start, install, and configure the vSRX, make sure your VM host meets the following recommended hardware, server platform, and software requirements as provided in Table 1.2.

Table 1.2 Minimum Hardware and Software Requirements

Requirements	Description
Linux KVM Hypervisor support	Ubuntu 14.04.5, 16.04, and 16.10
Memory	4-32 GB
Disk space	20 GB IDE drive
vCPUs	2-17 vCPUs
Network Interface Cards	2-8 vNICs Virtio SR-IOV (Intel 82599, X520/X540) SR-IOV (X710/XL710) PCI pass-through (Intel XL710). PCI pass-through (Intel XL710) is required if you intend to scale the performance and capacity of a vSRX to 9 or 17 vCPUs and 16 or 32 GB vRAM.
Software Bridges	Supports software-based virtual switches such as the Linux bridge or the OpenVswitch bridge, and direct connectivity to PCI Pass-through or an SR-IOV capable adapter.

MORE? For the latest updates to these requirements and the possible addition of more supported platforms, always check the TechLibrary first: https://www.juniper.net/documentation/en_US/vsrx/topics/reference/general/security-vsrx-system-requirement-with-kvm.html.

You may need to download a specific Junos OS release to take advantage of certain features.

vSRX Sizing Information

Table 1.3 lists the multicore vSRX *flavors* available for deployment.

Table 1.3 Available vSRX Flavors

Flavors	RE vCPUs	PFE vCPUs	vRAM
Small	1	1	4G
Medium	1	4	8G
Large	1	8	16G
Extra Large	1	16	32G

For example, if a vSRX VM has 2 vCPUs and 4 GB of vRAM, the vSRX boots to the smaller vCPU size. You can scale up a vSRX instance to a higher number of vCPUs and amount of vRAM, but you cannot scale down an existing vSRX instance to a smaller setting.

NOTE Scaling of the VM is discussed at the end of this book.

MORE? This *Day One* book is written with Junos OS (18.4) using vSRX3.0 architecture. In 18.4, vSRX3.0 supports small and medium flavors; support for higher flavors are planned in upcoming releases. Please check the latest version release notes for confirmation of the same:: https://www.juniper.net/documentation/en_US/vsrx/information-products/topic-collections/release-notes/18.4/index.html.

Obtaining a vSRX Evaluation License

Okay, before installing the vSRX, the last item on your checklist is whether you have an appropriate license. There's good news here.

To speed deployment of licensed features, the vSRX software image provides you with a 60-day product evaluation or trial license. This means when you download and install the vSRX image, you are entitled to use the trial license for 60 days. This product-unlocking license is required in order to use the basic functions of the

vSRX, such as networking, routing, and basic security features (such as stateful firewall). You need to install a 30-day advanced security features license in order to configure advanced security features.

DOWNLOAD You can download the trial license for advanced security features from the vSRX Free Trial License Page at: https://www.juniper.net/us/en/dm/free-vsrx-trial/.

Summary

Those are the fast track basics of virtualization and how a virtualized architecture and network are different than a traditional architecture and network. The chapter also covered the vSRX architecture (Routing Engine and Packet Forwarding Engine), the platforms vSRX supports, what BSD vSRX is based on, and how vSRX communicates within and outside a KVM host. And you have a checklist for the minimum requirements for running a vSRX instance and the evaluation license program. Now let's install vSRX on KVM.

Chapter 2

Getting Started with vSRX on KVM

Let's dig deeper into virtualization and create our first VM running the top-grade Junos OS. Upon completion of this chapter, you will have the host with all the required software packages to run a VM. The host will be running multiple virtual networks and a vSRX VM which you will configure and manage using basic commands supported on the Junos security platform. Let's get started.

Preparing the Host System for vSRX Installation

Chapter 1 discussed the architecture of vSRX. A *virtual flavor* of SRX can be instantiated on various platforms. In this *Day One* book, we will concentrate on KVM running on Ubuntu as the host operating system.

NOTE All configuration steps have been tested with Ubuntu versions 14.04.4 and 16.04. Though the snapshots in this *Day One* book are based on version 16.04, version 14.04 varies in naming convention of the interfaces (emX in 14.04 as compared to enoX in 16.04).

To install Ubuntu 16.04 in the host server, you need to download the ISO image from the Ubuntu website: http://releases.ubuntu.com/16.04/. The Ubuntu image available at the time of writing this book is: *Ubuntu-16.04.5-server-amd64.iso*. The ISO can be loaded as a Virtual CD and a boot sequence can be set to boot from the said ISO installation media. The installation process has multiple GUI steps that you need to follow to install the Ubuntu server.

IMPORTANT Follow the tutorial on the Ubuntu website: https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-server-1604#0 .

Okay. Upon installation of Ubuntu 16.04 on your host, let's verify the software and kernel version and install the required Linux packages.

Verifying Software and Kernel Version

Follow these steps on your host machine to check basic information.

Step 1: Log in to the host machine using the SSH connection.

Step 2: Get to know your host system better by learning the name of the host, software version, Linux kernel, and so on.

To check the details of the host, use the command `uname` (short for *UNIX name*) which prints the details of the host:

```
root@LabHost:~# uname
Linux
(uname defines the kernel of the host)
Append '--help' with uname and you will immediately see list of possible entries.
Options (-s,-r,-v,-p) provide the information we require or we can use -a to view all details.
root@LabHost:~# uname -s
Linux
root@LabHost:~# uname -r
4.4.0-131-generic
root@LabHost:~# uname -v
#157-Ubuntu SMP Thu Jul 12 15:51:36 UTC 2018
root@LabHost:~# uname -p
x86_64
root@LabHost:~# uname -a
Linux LabHost 4.4.0-131-generic #157-Ubuntu SMP Thu Jul 12 15:51:36 UTC 2018 x86_64 x86_64 x86_64 GNU/
Linux
```

Step 3: Check the Ubuntu version:

```
root@LabHost:~# cat /proc/version
Linux version 4.4.0-131-generic (buildd@lgw01-amd64-015) (gcc version 5.4.0 20160609 (Ubuntu
5.4.0-6ubuntu1~16.04.10) ) #157-Ubuntu SMP Thu Jul 12 15:51:36 UTC 2018
```

Installing Required Linux Packages

Once you have a host running with the Ubuntu operating system, the following steps will help you to confirm that all the required packages are up to date, and if they are not, to get them installed.

NOTE The apt in `apt-get` stands for *advanced packaging tool* and is a package manager that allows the Linux system to download and install the packages. The utility first checks the host for available packages, then updates the existing package by downloading the new files required to keep the package up to date. Follow these steps.

NOTE It is recommended that you log in as a root user so as not to use `sudo` in each command and enter `password` twice.

Step 1: Update the list of available packages and their versions.

```
root@LabHost:~# apt-get update
```

Step 2: Install latest versions of the packages you have.

```
root@LabHost:~# apt-get upgrade
```

Step 3: Install the KVM and other required packages.

```
root@labHost:~# apt-get install qemu-kvm libvirt-bin bridge-utils
```

Step 4: Install GUI for Linux, that is, `virt-manager`.

```
root@labHost:~# apt-get install virt-manager
```

Step 5: Install the QEMU system package.

```
root@labHost:~# apt-get install qemu-system
```

Let's divide an `apt-get` option into the following four stages:

- Checking the required and already installed packages.
- Fetching the required files.
- Unpacking the required files.
- Installing the package.

NOTE You must have HTTP access to the Internet to download these packages.

For example, here is what `apt-get install` output looks like:

```
root@LabHost:~# apt-get install qemu-system
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  qemu-slof qemu-system-arm qemu-system-mips qemu-system-misc qemu-system-ppc qemu-system-sparc
Suggested packages:
  qemu samba vde2 openbios-ppc openhackware
The following NEW packages will be installed:
  qemu-slof qemu-system qemu-system-arm qemu-system-mips qemu-system-misc qemu-system-ppc qemu-
system-sparc
0 upgraded, 7 newly installed, 0 to remove and 4 not upgraded.
Need to get 23.7 MB of archives.
After this operation, 154 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://in.archive.ubuntu.com/ubuntu xenial-updates/main amd64 qemu-system-
arm amd64 1:2.5+dfsg-5ubuntu10.34 [4,120 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu xenial-updates/main amd64 qemu-system-
mips amd64 1:2.5+dfsg-5ubuntu10.34 [4,924 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu xenial-updates/main amd64 qemu-slof all 20151103+dfsg-
```

```
1ubuntu1.1 [173 kB]
Get:4 http://in.archive.ubuntu.com/ubuntu xenial-updates/main amd64 qemu-system-
ppc amd64 1:2.5+dfsg-5ubuntu10.34 [5,747 kB]
Get:5 http://in.archive.ubuntu.com/ubuntu xenial-updates/main amd64 qemu-system-
sparc amd64 1:2.5+dfsg-5ubuntu10.34 [2,000 kB]
Get:6 http://in.archive.ubuntu.com/ubuntu xenial-updates/main amd64 qemu-system-
misc amd64 1:2.5+dfsg-5ubuntu10.34 [6,773 kB]
Get:7 http://in.archive.ubuntu.com/ubuntu xenial-updates/main amd64 qemu-system amd64 1:2.5+dfsg-
5ubuntu10.34 [6,104 B]
Fetched 23.7 MB in 2min 55s (136 kB/s)
Selecting previously unselected package qemu-system-arm.
(Reading database ... 84651 files and directories currently installed.)
Preparing to unpack .../qemu-system-arm_1%3a2.5+dfsg-5ubuntu10.34_amd64.deb ...
Unpacking qemu-system-arm (1:2.5+dfsg-5ubuntu10.34) ...
Selecting previously unselected package qemu-system-mips.
Preparing to unpack .../qemu-system-mips_1%3a2.5+dfsg-5ubuntu10.34_amd64.deb ...
Unpacking qemu-system-mips (1:2.5+dfsg-5ubuntu10.34) ...
Selecting previously unselected package qemu-slof.
Preparing to unpack .../qemu-slof_20151103+dfsg-1ubuntu1.1_all.deb ...
Unpacking qemu-slof (20151103+dfsg-1ubuntu1.1) ...
Selecting previously unselected package qemu-system-ppc.
Preparing to unpack .../qemu-system-ppc_1%3a2.5+dfsg-5ubuntu10.34_amd64.deb ...
Unpacking qemu-system-ppc (1:2.5+dfsg-5ubuntu10.34) ...
Selecting previously unselected package qemu-system-sparc.
Preparing to unpack .../qemu-system-sparc_1%3a2.5+dfsg-5ubuntu10.34_amd64.deb ...
Unpacking qemu-system-sparc (1:2.5+dfsg-5ubuntu10.34) ...
Selecting previously unselected package qemu-system-misc.
Preparing to unpack .../qemu-system-misc_1%3a2.5+dfsg-5ubuntu10.34_amd64.deb ...
Unpacking qemu-system-misc (1:2.5+dfsg-5ubuntu10.34) ...
Selecting previously unselected package qemu-system.
Preparing to unpack .../qemu-system_1%3a2.5+dfsg-5ubuntu10.34_amd64.deb ...
Unpacking qemu-system (1:2.5+dfsg-5ubuntu10.34) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up qemu-system-arm (1:2.5+dfsg-5ubuntu10.34) ...
Setting up qemu-system-mips (1:2.5+dfsg-5ubuntu10.34) ...
Setting up qemu-slof (20151103+dfsg-1ubuntu1.1) ...
Setting up qemu-system-ppc (1:2.5+dfsg-5ubuntu10.34) ...
Setting up qemu-system-sparc (1:2.5+dfsg-5ubuntu10.34) ...
Setting up qemu-system-misc (1:2.5+dfsg-5ubuntu10.34) ...
Setting up qemu-system (1:2.5+dfsg-5ubuntu10.34) ...
```

Virtual Networks

You need virtualized networks to process packets between VMs.

Our host OS is Ubuntu, and it is presumed you have installed KVM. We have KVM convert the host OS into a hypervisor and expose the underlining hardware to the VM. The VMs have vNIC and one VM can have multiple vNICs. The host NIC is called a *pNIC* (physical NIC).

Virtual networks can be broadly classified as:

1. Linux bridge
2. OpenvSwitch [Out of scope of this *Day One* +book]

A Linux bridge acts as a network switch. You can connect both physical interfaces (example: eth0) and virtual interfaces to the Linux bridge.

An OpenvSwitch (OVS) is an open source multilayer virtual switch. It enables massive network automation through programmatic extensions. It can replace Linux bridges.

Both Linux bridge and OVS offer switching infrastructure for the VMs to communicate. Also, pNICs can be connected to either for out-of-host connectivity.

This *Day One* focuses on using Linux bridges for VM communication.

With KVM installed correctly on the host, a predefined network named "default" is already configured for us. Follow these steps on your host to verify that the default network is created:

Step 1: Check the networks installed on the host.

```
root@LabHost:~# virsh net-list --all
root@LabHost:/etc/libvirt/qemu/networks# virsh net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
```

Step 2: Check the details of the networks.

To display the details of the default network in XML format, use the option `more` as shown here:

```
root@LabHost:~# more /etc/libvirt/qemu/networks/default.xml
<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made
using:
  virsh net-edit default
or other application using the libvirt API.
-->

<network>
  <name>default</name>
  <uuid>908e88b6-2f5a-40ba-843e-09a5286ad764</uuid>
  <forward mode='nat'/>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:56:10:fc'/>
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
</network>
root@LabHost:~#
```

Step 3: Display more details about the default installed network.

```
root@LabHost:~# virsh net-dumpxml default
<network>
  <name>default</name>
  <uuid>908e88b6-2f5a-40ba-843e-09a5286ad764</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:56:10:fc'/>
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'/>
    </dhcp>
  </ip>
</network>
```

The output here shows that the network `default` is part of the Linux Bridge `virbr0`.

Step 4: Check the details of network using bridge utility `brctl`:

```
root@LabHost:~# brctl show virbr0
root@LabHost:~# brctl show virbr0
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.5254005610fc       yes             virbr0-nic
```

Configuring Virtual Networks

Now let's create some virtual networks to bind the VMs we create together. Create the following three virtual networks:

1. Default
2. Routed [green_net]
3. Routed [red_net]

Create the Management [Default] Virtual Network

If the host does not have the default network configured, use the following steps to create a default network. Otherwise, skip this procedure and start creating the green_net and red_net networks.

IMPORTANT If the host already has a default network and if you try to create it again, you might get an error message.

NOTE The networks are written in XML format and the virsh utility stores the XML files of the networks in /etc/libvirt/qemu/networks.

Step 1: Navigate to the directory: /etc/libvirt/qemu/networks:

```
root@LabHost:~# cd /etc/libvirt/qemu/networks
```

Step 2: Create the XML file for the network. Copy and paste the following snippet:

```
root@LabHost: /etc/libvirt/qemu/networks # nano default.xml
<network>
  <name>default</name>
  <bridge name="virbr0"/>
  <forward mode='nat'/>
  <nat>
    <port start ='1024' end='65535' />
  </nat>
  <bridge name='virbr0' stp='on' delay='0' />
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2"end="192.168.122.254"/>
    </dhcp>
   </ip>
 </network>
```

Step 3: Press Ctrl-X to exit and Press Y for Yes to save the changes.

Step 4: Define, start, and set the network to `autostart` once boot process completes.

Use the following three commands to first define, then start/autostart, the network:

```
root@LabHost:~# virsh net-define /etc/libvirt/qemu/networks/default.xml
root@LabHost:~# virsh net-start default
root@LabHost:~# virsh net-autostart default
```

TIP The default network is by default set in "NAT" mode, meaning the management interface is connected to the Internet, and the VM can also have Internet connectivity as the private IP address would be NATED automatically by the Host.

Create the Routed [green_net] Virtual Network

Let's use the following steps to create a network green_net.

Step 1: Navigate to the location and create XML file. Create a file with the name `green_net.xml` at `/etc/libvirt/qemu/networks`. Enter the following snippet:

```
root@LabHost:~# nano /etc/libvirt/qemu/networks/green_net.xml
<network>
  <name>green_net</name>
  <forward mode='route'/>
  <bridge name='green_net' stp='on' delay='0'/>
  <ip address='192.168.123.1' netmask='255.255.255.0'>
    <dhcp>
     <range start='192.168.123.100' end='192.168.123.250'/>
    </dhcp>
  </ip>
</network>
```

Step 2: Press Ctrl-X to exit and Press Y for Yes to save the changes.

Step 3: Define, start, and set the network to autostart once the boot process completes. Use the following three commands to first define the network, and then start/autostart the network:

```
root@LabHost:~# virsh net-define /etc/libvirt/qemu/networks/green_net.xml
root@LabHost:~# virsh net-start green_net
root@LabHost:~# virsh net-autostart green_net
root@LabHost:~# virsh net-define /etc/libvirt/qemu/networks/green_net.xml
Network green_net defined from /etc/libvirt/qemu/networks/green_net.xml

root@LabHost:~# virsh net-start green_net
Network green_net started

root@LabHost:~# virsh net-autostart green_net
Network green_net marked as autostarted
```

The network is created and started. The XML file should be updated with the unique UUID (universally unique identifier) and MAC address.

Step 4: Check the XML for details. Open the file green_net.xml that you just created and check the changes:

```
root@LabHost:~# more /etc/libvirt/qemu/networks/green_xml.xml
root@LabHost:~# more /etc/libvirt/qemu/networks/green_net.xml
<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made
using:
  virsh net-edit green_net
or other application using the libvirt API.
-->

<network>
  <name>green_net</name>
  <uuid>cc8867f9-4523-4f1f-b8b6-5e19c36084fe</uuid>
  <forward mode='route'/>
  <bridge name='green_net' stp='on' delay='0'/>
  <mac address='52:54:00:4b:9a:07'/>
  <ip address='192.168.123.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.123.100' end='192.168.123.250'/>
    </dhcp>
  </ip>
</network>
root@LabHost:~#

root@LabHost:~# virsh net-list
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
 green_net            active     yes           yes
```

NOTE Since this virtual network is in `route` mode, traffic would only be routed and not NAT'd. Also note the IP address is useful if DHCP is being used for the connected interfaces, or else the interfaces can be connected with any static address and this virtual network will work as a normal bridge.

Create the Routed [red_net] Virtual Network

Use the following steps to create a network green_net:

Step 1: Create an XML file in directory: /etc/libvirt/qemu/networks. You can use the following snippet:

```
root@LabHost:~# nano /etc/libvirt/qemu/networks/red_net.xml
<network>
  <name>red_net</name>
  <forward mode='route'/>
  <bridge name='red_net' stp='on' delay='0'/>
  <ip address='192.168.124.1' netmask='255.255.255.0'>
    <dhcp>
     <range start='192.168.124.100' end='192.168.124.250'/>
    </dhcp>
  </ip>
</network>
```

Step 2: Define, start, and auto-start the network:

```
root@LabHost:~# virsh net-define /etc/libvirt/qemu/networks/red_net.xml
Network red_net defined from /etc/libvirt/qemu/networks/red_net.xml
root@LabHost:~# virsh net-start red_net
Network red_net started
root@LabHost:~# virsh net-autostart red_net
Network red_net marked as autostarted
```

Step 3: Check that the network has been started:

```
root@LabHost:~# more /etc/libvirt/qemu/networks/red_net.xml
<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made using:
  virsh net-edit red_net
or other application using the libvirt API.
-->

<network>
  <name>red_net</name>
  <uuid>3ef5070e-1cf2-4095-a2a8-c162003fdb87</uuid>
  <forward mode='route'/>
  <bridge name='red_net' stp='on' delay='0'/>
  <mac address='52:54:00:16:2d:c5'/>
  <ip address='192.168.124.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.124.100' end='192.168.124.250'/>
    </dhcp>
  </ip>
</network>
```

Next, let's take another look at the three networks that we have created. Run the following two commands to get network information.

Virsh command:

```
root@LabHost:~# virsh net-list --all
root@LabHost:~# virsh net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
 green_net            active     yes           yes
 red_net              active     yes           yes
```

Brctl command:

```
root@LabHost:~# brctl show
root@LabHost:~# brctl show
bridge name     bridge id               STP enabled     interfaces
green_net       8000.5254004b9a07       yes             green_net-nic
red_net         8000.525400162dc5       yes             red_net-nic
virbr0          8000.5254005610fc       yes             virbr0-nic
```

Installing vSRX on KVM

Now that you have the required packages, such as KVM, QEMU, and Libvirt, installed on the host OS, and you've created a few virtual networks, too, it's time to instantiate our first vSRX VM.

A VM instance requires specifying multiple important parameters that define a running VM. Let's jot down all the required ones. Values shown in the following list are sample values only and you must change any details necessary to match your network configuration:

- VM name [=vSRX_00]
- RAM [=4096]
- CPU Model [=SandyBridge]
- Number of vCPUs [=2]
- Base Architecture [=x86_64]
- Image of the VM [=<vSRX>.qcow2]
- Image Format [=qcow2]
- Disk Size [=20]
- Device Type [=disk]
- OS Type [=Linux]
- Networks to be imported [=default, green_net,red_net]

Use the CLI and the `virt-install` command to pass all the required parameters and start the installation process for a vSRX instance.

NOTE A GUI version (virt-manager) is also available to perform the same task. However, some actions cannot be performed using the GUI and you are required to use the CLI. So, to avoid confusion, we have used CLI throughout the book and discussed the GUI procedure in the Appendix.

To create the VM, use the following sequence.

Step 1: Download your copy for the vSRX from the Juniper website: https://support.juniper.net/support/downloads/.

Step 2: Now, retain this image file as a master file and also create a copy of the image file.

When you create a copy of the image file, name it in-line with the VM you are about to spin. In this example, copy and name the image file as `img_vSRX_00.qcow2;` and then use the name for creating vSRX_00 VM:

```
root@LabHost:/var/lib/libvirt/images# cp junos-vsrx3-x86-64-18.4R1.8.qcow2 img_vSRX_00.qcow2
root@LabHost:/var/lib/libvirt/images# cd ~
root@LabHost:~# ls -la /var/lib/libvirt/images/
total 1430284
drwx--x--x 2 root root      4096 Jan 26 08:38 .
drwxr-xr-x 7 root root      4096 Jan 26 06:36 ..
-rw-r--r-- 1 root root 732299264 Jan 26 08:38 img_vSRX_00.qcow2
-rw-r--r-- 1 root root 732299264 Jan 26 05:22 junos-vsrx3-x86-64-18.4R1.8.qcow2
```

Step 3: Install the vSRX VM.

Copy and paste the following snippet in a text editor to confirm that spaces are copied correctly:

```
virt-install --name vSRX_00 --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/
lib/libvirt/images/img_vSRX_00.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:green_
net,model=virtio --network=network:red_net,model=virtio
root@LabHost:~# virt-
install --name vSRX_00 --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/
libvirt/images/img_vSRX_00.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:green_
net,model=virtio --network=network:red_net,model=virtiok=network:green_
net,model=virtio --network=network:red_net,model=virtio

Starting install...
Creating domain...                                                    |    0 B  00:00:04

(virt-viewer:3196): GSpice-WARNING **: PulseAudio context failed Connection refused

(virt-viewer:3196): GSpice-WARNING **: pa_context_connect() failed: Connection refused
Domain creation completed.
root@LabHost:~#
```

NOTE Upon executing this command the virt-viewer console window will open, which shows the boot logs printed on its screen. Close the window to complete the domain creation.

Step 4: Check the status of the installed VM using the `virsh` command:

```
root@LabHost:~# virsh list –all
root@LabHost:~# virsh list ––all
 Id    Name                           State
----------------------------------------------------
 1     vSRX_00                        running
The output defines the unique identifier, name and the state of the VM.
```

Step 5: Check if the virtual networks you specified as parameters have been connected to the VM:

```
root@LabHost:~# virsh domiflist vSRX_00
root@LabHost:~# virsh domiflist vSRX_00
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      network    default    virtio      52:54:00:86:82:c2
vnet1      network    green_net  virtio      52:54:00:25:e1:06
vnet2      network    red_net    virtio      52:54:00:9f:5e:6e
```

The output here displays that the specified networks are part of the VM, and also lists details of the assigned MAC address and vNIC interface on the VM side.

Managing vSRX VM on KVM

From the host, you can directly connect to the vSRX instance using the `virsh` command. Use the following steps to manage the VM:

Step 1: Access vSRX VM. Type the following `virsh` command to connect to the console of the VM:

```
root@LabHost:~# virsh console vSRX_00
root@LabHost:~# virsh console vSRX_00
Connected to domain vSRX_00
Escape character is ^]
lag enhanced disabled 0
 <….>
```

If you execute the `virsh console vSRX_00` command right after the `virt-install`, you can watch the progress of the installation; it's a quick process if you are using vSRX version for Junos OS Release 18.4. But if you log on to the console after boot the process completes, you'll see the following prompt:

```
root@LabHost:~# virsh console vSRX_00
Connected to domain vSRX_00
Escape character is ^]
```

From here, you need to press Return [Enter Keyword] to get the login prompt:

```
root@LabHost:~# virsh console vSRX_00
Connected to domain vSRX_00
Escape character is ^]

FreeBSD/amd64 (Amnesiac) (ttyu0)
login:
```

Step 2: At the login prompt, enter the root and at the password prompt, press Enter:

```
root@LabHost:~# virsh console vSRX_00
Connected to domain vSRX_00
Escape character is ^]
lag enhanced disabled 0

FreeBSD/amd64 (Amnesiac) (ttyu0)

login: root

--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@:~ #
```

Step 3: After you are authenticated, verify or check the vSRX with following commands. To get to the CLI from the shell prompt, enter `cli` and then enter the `show version` command to confirm the version of the VM:

```
root@:~ # cli
root>

root> show version
Model: vSRX
Junos: 18.4R1.8
JUNOS OS Kernel 64-bit XEN [20181207.6c2f68b_2_builder_stable_11]
JUNOS OS libs [20181207.6c2f68b_2_builder_stable_11]
JUNOS OS runtime [20181207.6c2f68b_2_builder_stable_11]
JUNOS OS time zone information [20181207.6c2f68b_2_builder_stable_11]
JUNOS OS libs compat32 [20181207.6c2f68b_2_builder_stable_11]
JUNOS OS 32-bit compatibility [20181207.6c2f68b_2_builder_stable_11]
JUNOS py extensions [20181217.004159_builder_junos_184_r1]
JUNOS py base [20181217.004159_builder_junos_184_r1]
```

Step 4: Check the hardware version:

```
root> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                8287c40d8c2b      VSRX
Midplane
System IO
Routing Engine                                            VSRX-S
FPC 0                                                     FPC
  PIC 0                                                   VSRX DPDK GE
Power Supply 0
```

Note here that for Routing Engine, VSRX-S means that this is a *small* flavor of vSRX.

Step 5: Check the Packet Forwarding Engine status:

```
root> show chassis fpc pic-status
Slot 0   Online       FPC
  PIC 0  Online       VSRX DPDK GE
```

Step 6: Check the interfaces that are available for configuration:

```
root> show interfaces terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
gr-0/0/0                up    up
ip-0/0/0                up    up
lsq-0/0/0               up    up
lt-0/0/0                up    up
mt-0/0/0                up    up
sp-0/0/0                up    up
sp-0/0/0.0              up    up   inet
                                   inet6
sp-0/0/0.16383          up    up   inet
ge-0/0/1                up    up
dsc                     up    up
fti0                    up    up
fxp0                    up    up
fxp0.0                  up    up
gre                     up    up
ipip                    up    up
irb                     up    up
lo0                     up    up
lo0.16384               up    up   inet     127.0.0.1           --> 0/0
lo0.16385               up    up   inet     10.0.0.1            --> 0/0
                                            10.0.0.16           --> 0/0
                                            128.0.0.1           --> 0/0
                                            128.0.0.4           --> 0/0
                                            128.0.1.16          --> 0/0
lo0.32768               up    up
lsi                     up    up
mtun                    up    up
pimd                    up    up
pime                    up    up
pp0                     up    up
ppd0                    up    up
ppe0                    up    up
st0                     up    up
tap                     up    up
vlan                    up    down
```

Table 2.4 lists the virtual networks mapped to the interfaces on vSRX VM.

Table 2.4 Network to Interface Mapping

Network Name	vSRX Interfaces
default	fxp0
green_net	ge-0/0/0
red_net	ge-0/0/1

NOTE The order in which networks are added using the `virt-install` command determines its numbering in the VM.

Step 7: Log out from the VM and get back to the host, (press Ctrl +) from the keyboard to return to the host).

Step 8: Stop the VM. To stop a VM gracefully, first perform a power off from Junos:

```
root> request system power-off
```

Next, from the host, execute the following so it does not delete the VM instance but just stops the VM:

```
root@LabHost:~# virsh destroy vSRX_00
root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 1     vSRX_00                        running
root@LabHost:~# virsh destroy vSRX_00
Domain vSRX_00 destroyed

root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 -     vSRX_00                        shut off
```

Step 9: To restart or power up the inactive vSRX instance, execute the following `virsh` command:

```
root@LabHost:~# virsh start vSRX_00
root@LabHost:~# virsh start vSRX_00
Domain vSRX_00 started

root@LabHost:~#
root@LabHost:~# virsh console vSRX_00
Connected to domain vSRX_00
Escape character is ^]
/packages/sets/active/boot/os-kernel/
kernel text=0x451f38 data=0x83b38+0x30d940 syms=[0x8+0x95f28+0x8+0x826f2]
/packages/sets/active/boot/junos-net-platform/mtx_re.ko size 0x284fd8 at 0xcfc000
loading required module 'netstack'
/packages/sets/active/boot/netstack/netstack.ko size 0x1496958 at 0xf81000
loading required module 'crypto'
/packages/sets/active/boot/os-crypto/crypto.ko size 0x43df0 at 0x2418000
loading required module 'pvi_db'

root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     vSRX_00                        running
```

Configuring vSRX VM on KVM

Since you have installed your first VM instance, let's start configuring it by issuing a few basic commands in the following steps.

Step 1: Navigate through different modes in CLI. The '>' symbol shows that we are in operational mode. Type the following command to enter into configuration mode:

```
root> configure
[edit]
root#
'#' sign with edit in square brackets defines the configuration mode.
```

Step 2: Now configure the credentials and hostname. To set the root password type the following command and then enter the password twice:

```
[edit]
root# set system root-authentication plain-text-password
New password: <Type Once>
ReType New Password : <Type Again>
root> configure
Entering configuration mode

[edit]
root# set system root-authentication plain-text-password
New password:
Retype new password:

[edit]
root# set system host-name vSRX_00

[edit]
root# commit
commit complete

[edit]
root@vSRX_00#
```

Step 3: Enter into configuration mode (working with the # hashtag prompt) once Junos responds `commit complete`, this confirms that the configuration has been applied. Notice that we are in configuration mode.

Step 4: Navigating through the different modes, exit three times so as to log in with the just-set root password:

```
[edit]
root@vSRX_00# exit
Exiting configuration mode

root@vSRX_00> exit
root@:~ # exit
logout
```

```
FreeBSD/amd64 (vSRX_00) (ttyu0)
login: root
Password:
Last login: Sat Jan 26 03:23:47 on ttyu0

--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@vSRX_00:~ #
```

Since only one console connection to the VM is allowed, what if you need multiple console connections for multiple sessions to the same VM?

The solution is to use SSH connections to the VM from the host on the fxp0 (management). Remember, while configuring/checking the "default" predefined VM, we saw an IP subnet defined and a DHCP address space allocated [192.168.122.2 to 192.168.122.254]. We can leverage the same and configure the vSRX fxp interface to act as a DHCP client to receive an IP address in the range.

To gain SSH access to the VM follow these steps.

Step 1: Log in to vSRX_00VM using the console and configure fxp0:

```
[edit]
root@vSRX_00# set interfaces fxp0.0 family inet dhcp-client

[edit]
root@vSRX_00# commit and-quit
commit complete
Exiting configuration mode
```

Step 2: Check to see that the IP address is assigned:

```
root@vSRX_00> show interfaces terse | match fxp
fxp0                    up    up
fxp0.0                  up    up   inet     192.168.122.144/24

root@vSRX_00>
```

Step 3: Log out from the console and try to log in:

```
root@LabHost:~# ssh 192.168.122.144
The authenticity of host '192.168.122.144 (192.168.122.144)' can't be established.
ECDSA key fingerprint is SHA256:xfvj3h7Ee2Ji+TB0nWIXdpdkYEjsqNLHYt5k8UanMbg.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.122.144' (ECDSA) to the list of known hosts.
Password:
Password:
Password:
Received disconnect from 192.168.122.144 port 22:2: Too many password failures for root
Connection to 192.168.122.144 closed by remote host.
Connection to 192.168.122.144 closed.
root@LabHost:~#
```

There seems to be a problem. We are trying to log in using SSH with root, which requires it to be explicitly allowed in the configuration.

Step 4: Allow the SSH root access. This configuration allows users to log in to the VM as `root` through SSH:

```
[edit]
root@vSRX_01# set system services ssh root-login allow

[edit]
root@vSRX_01# commit
commit complete
```

Step 5: Log out from VM and retry SSH from the host:

```
root@LabHost:~# ssh 192.168.122.144
Password:
Last login: Tue Jan 29 11:44:17 2019
--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@vSRX_01:~ #
```

Checking Licenses Installed

The following sample shows details of an evaluation license in the CLI:

```
root@vSRX_00> show system license
License usage:
                                Licenses     Licenses    Licenses    Expiry
  Feature name                       used    installed      needed
  Virtual Appliance                     1            1           0    59 days
  remote-access-ipsec-vpn-client        0            2           0    permanent
Licenses installed:
  License identifier: E420588955
  License version: 4
  Software Serial Number: 20150625
  Customer ID: vSRX-JuniperEval
  Features:
    Virtual Appliance - Virtual Appliance
      count-down, Original validity: 60 days
root@vSRX_00>
```

Summary

You should now have all the information you need on how to build and manage a vSRX VM and connect it to the instance for lab purposes and other uses. That being said, Chapter 3 is all the more engaging because you are going to set up a few topologies and scale up an existing vSRX VM.

Chapter 3

Building a Simple Topology

Hey, congratulations on installing your first vSRX VM.

This chapter provides hands-on instruction to building small topologies that can be used as templates for building larger and more complex topologies. First it reviews the single vSRX VM that you created in Chapter 2, builds another vSRX instance, and verifies the communication between them. Then it creates a high availability cluster with two new VM instances, followed by stitching the first two topologies together using another instance acting as an Internet Router. You will then build a topology to understand how a vSRX VM can interact with the physical NIC using Linux bridge (virtual network), concluding with a topology where you bypass the Linux Bridge and connect the VM directly to the physical NIC using SR-IOV.

It's a busy chapter so let's get started!

Building Your First Topology

This lab creates a simple topology by using two vSRX instances as in Figure 3.1.

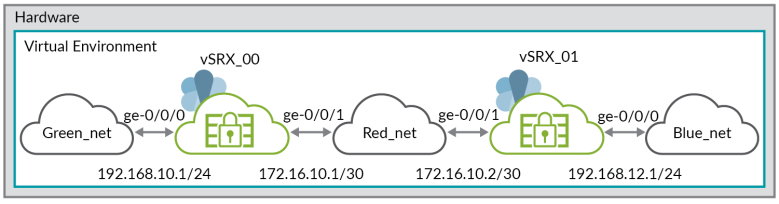


Figure 3.1 *Lab Topology for Site-to-Site Setup*

What do we already have with us? vSRX_00 VM. Okay, let's use that VM and see what we need to add more.

As per the first topology, the two VMs connect to each other using the red network (simulating WAN side) and the green and blue networks simulating LAN side for respective VM. If you recall, in Chapter 2 we created two virtual networks red_net and green_net. We shall be using the same networks and will create one more network for the vSRX_01 VM-side LAN connection.

The ge-0/0/0 interface from each VM is connected to green_net and blue_net networks and ge-0/0/1 interface is connected to the red_net network.

Our goals for this lab exercise are:

- Create the network as shown in the topology
- Configure the green_net, blue_net and red_net facing interfaces on vSRX VMs (vSRX_00 and vSRX_01)
- Ping from vSRX_00 to vSRX_01, via red_net-side interface
- Ping from vSRX_00 green_net interface to vSRX_01 blue_net interface

The following steps explore how to build your first vSRX topology on KVM.

Step 1: What we have is vSRX_00, let's check the status of the VM and the networks connected:

```
root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     vSRX_00                        running

root@LabHost:~# virsh domiflist vSRX_00
Interface  Type       Source     Model       MAC
-------------------------------------------------------------
vnet0      network    default    virtio      52:54:00:86:82:c2
vnet1      network    green_net  virtio      52:54:00:25:e1:06
vnet2      network    red_net    virtio      52:54:00:9f:5e:6e
```

TIPS "Domiflist" can be broken into a domain interface list to memorize.

NOTE Command "virsh domiflist <vm-name>" provides information about the virtual interface a VM is connected to and its details.

Step 2: Before creating the second VM, you need to create the virtual network blue_net.

Create a file in /etc/libvirt/qemu/networks with name vlue_net.xml. Copy and paste the following snippet:

```
root@LabHost:~# nano /etc/libvirt/qemu/networks/blue_net.xml
<network>
  <name>blue_net</name>
  <forward mode='route'/>
  <bridge name='blue_net' stp='on' delay='0'/>
  <ip address='192.168.125.1' netmask='255.255.255.0'>
    <dhcp>
     <range start='192.168.125.100' end='192.168.125.250'/>
    </dhcp>
  </ip>
</network>
```

If using nano, Press ^X and enter yes to save changes:

```
root@LabHost:~# more /etc/libvirt/qemu/networks/blue_net.xml
<network>
  <name>blue_net</name>
  <forward mode='route'/>
  <bridge name='blue_net' stp='on' delay='0'/>
  <ip address='192.168.125.1' netmask='255.255.255.0'>
    <dhcp>
     <range start='192.168.125.100' end='192.168.125.250'/>
    </dhcp>
  </ip>
</network>
```

Step 3: Define, start, and autostart the blue_net network:

```
root@LabHost:~# virsh net-define /etc/libvirt/qemu/networks/blue_net.xml
Network blue_net defined from /etc/libvirt/qemu/networks/blue_net.xml

root@LabHost:~# virsh net-start blue_net
Network blue_net started

root@LabHost:~# virsh net-autostart blue_net
Network blue_net marked as autostarted

root@LabHost:~#
```

Step 4: Confirm that the network is installed and started:

```
root@LabHost:~# virsh net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 blue_net             active     yes           yes
 default              active     yes           yes
 green_net            active     yes           yes
 red_net              active     yes           yes
```

Step 5: Next, let's instantiate the second VM, that is, vSRX_01. Navigate to the vSRX image location and create a copy as "img_vSRX_01.qcow2":

```
root@LabHost:~# cp /var/lib/libvirt/images/junos-vsrx3-x86-64-18.4R1.8.qcow2 img_vSRX_01.qcow2
root@LabHost:~#
root@LabHost:~# ls -la /var/lib/libvirt/images/
total 2274832
drwx--x--x 2 root         root      4096 Jan 27 15:56 .
```

```
drwxr-xr-x 7 root         root      4096 Jan 26 06:36 ..
-rw-r--r-- 1 libvirt-qemu kvm  864813056 Jan 27 15:56 img_vSRX_00.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 27 15:56 img_vSRX_01.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 26 05:22 junos-vsrx3-x86-64-18.4R1.8.qcow2
```

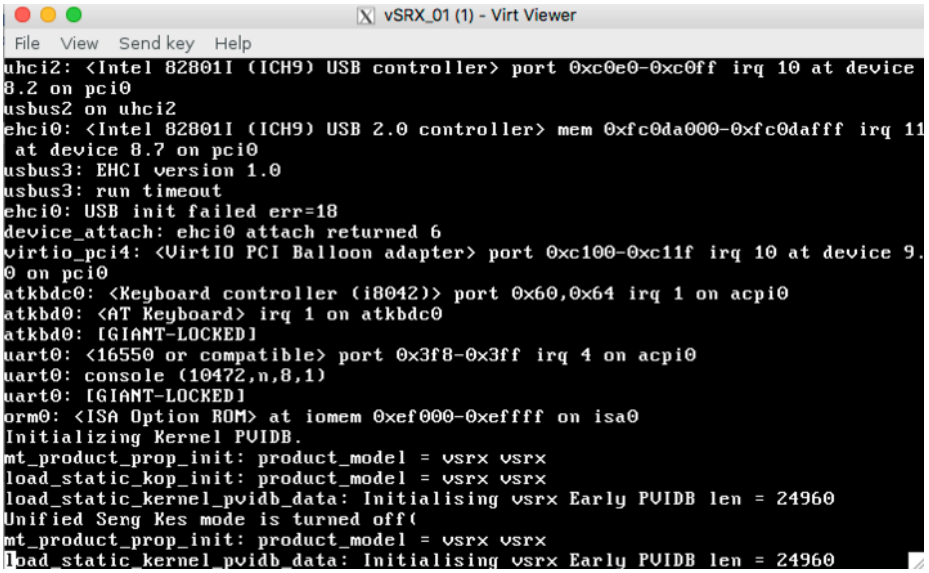
Step 6: Use the virsh command to install vSRX_01. When running multiple instances of vSRX on the same host, each vSRX instance needs to be configured with a unique identifier:

```
root@LabHost:~# virt-
install --name vSRX_01 --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/
libvirt/images/img_vSRX_01.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:blue_
net,model=virtio --network=network:red_net,model=virtio

root@LabHost:~# virt-
install --name vSRX_01 --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/
libvirt/images/img_vSRX_01.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:blue_
net,model=virtio --network=network:red_net,model=virtio

Starting install...
Creating domain...                                                                    |
0 B  00:00:04
(virt-viewer:9510): GSpice-WARNING **: PulseAudio context failed Connection refused

(virt-viewer:9510): GSpice-WARNING **: pa_context_connect() failed: Connection refused
Domain creation completed.
```



Step 7: Check the status and the interface connected using the following commands:

```
root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     vSRX_00                        running
 3     vSRX_01                        running
root@LabHost:~# virsh domiflist vSRX_01
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet3      network    default    virtio      52:54:00:37:a4:bd
vnet4      network    blue_net   virtio      52:54:00:67:34:eb
vnet5      network    red_net    virtio      52:54:00:90:c9:18
```

Step 8: Check the overview of the virtual networks and interfaces:

```
root@LabHost:~# brctl show
bridge name     bridge id               STP enabled     interfaces
blue_net        8000.5254002090c0       yes             blue_net-nic
                                                        vnet4
green_net       8000.5254004b9a07       yes             green_net-nic
                                                        vnet1
red_net         8000.525400162dc5       yes             red_net-nic
                                                        vnet2
                                                        vnet5
virbr0          8000.5254005610fc       yes             virbr0-nic
                                                        vnet0
                                                        vnet3
```

Step 9: Connect to console of vSRX_01 and configure the root password and the hostname:

```
root@LabHost:~# virsh console vSRX_01
Connected to domain vSRX_01
Escape character is ^]

FreeBSD/amd64 (Amnesiac) (ttyu0)
login: root

--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@:~ # cli
root> configure
Entering configuration mode
[edit]
root# set system root-authentication plain-text-password
New password:
Retype new password:

[edit]
root# set system host-name vSRX_01

[edit]
root# commit
commit complete

[edit]
root@vSRX_01#
```

Now that the two vSRX VMs are up and running with the connected virtual networks, per the requirement, you should be able to:

- Ping to vSRX_01 red_net interface from vSRX_00 red_net interface.
- Ping to vSRX_01 blue_net interface from vSRX_00 green_net interface, over the red_net interface.

Let's revisit the interface to network mapping as shown in Table 3.1.

Table 3.1 Interface to Network Mapping

Network	vSRX Interface
default	fxp0
green_net/blue_net	ge-0/0/0
red_net	ge-0/0/1

The first interface mapped in `virt-install` command was green_net/blue_net and the same interface is mapped to the first revenue interface on the vSRX VM, that is, ge-0/0/0 in respective VM. To configure the vSRX_00 and vSRX_01, use the following steps.

Step 1: On vSRX_00, enter configuration mode, and complete the configuration for basic security zones and bind them to the traffic interfaces:

```
root@vSRX_00> configure
Entering configuration mode

[edit]
root@vSRX_00#
set security zones security-zone green host-inbound-traffic system-services all
set security zones security-zone green host-inbound-traffic protocols all
set security zones security-zone green interfaces ge-0/0/0.0
set security zones security-zone red host-inbound-traffic system-services all
set security zones security-zone red host-inbound-traffic protocols all
set security zones security-zone red interfaces ge-0/0/1.0
set interfaces ge-0/0/0 unit 0 family inet address 192.168.10.1/24
set interfaces ge-0/0/1 unit 0 family inet address 172.16.10.1/30
```

Step 2: On vSRX_01, enter configuration mode, and complete the configuration for basic security zones and bind them to the traffic interfaces:

```
root@vSRX_01> configure
Entering configuration mode

[edit]
root@vSRX_01#
set security zones security-zone blue host-inbound-traffic system-services all
set security zones security-zone blue host-inbound-traffic protocols all
set security zones security-zone blue interfaces ge-0/0/0.0
set security zones security-zone red host-inbound-traffic system-services all
set security zones security-zone red host-inbound-traffic protocols all
```

```
set security zones security-zone red interfaces ge-0/0/1.0
set interfaces ge-0/0/0 unit 0 family inet address 192.168.12.1/24
set interfaces ge-0/0/1 unit 0 family inet address 172.16.10.2/30
```

Now that you have configured the link between vSRX_00 VM and vSRX_01 VM using the interfaces ge-0/0/0 and ge-0/0/1, you need to check the connectivity over the interfaces.

Step 3: Validate the connectivity using the ping command on the red_net, vSRX_00 to vSRX_01 and vice-versa:

```
[edit]
root@vSRX_00# run ping 172.16.10.2 count 3
PING 172.16.10.2 (172.16.10.2): 56 data bytes
64 bytes from 172.16.10.2: icmp_seq=0 ttl=64 time=2.393 ms
64 bytes from 172.16.10.2: icmp_seq=1 ttl=64 time=0.833 ms
64 bytes from 172.16.10.2: icmp_seq=2 ttl=64 time=0.822 ms

--- 172.16.10.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.822/1.349/2.393/0.738 ms
```

Step 4: Further, validate the connectivity from green_net on vSRX_00 to blue_net interface IP on vSRX_01:

```
root@vSRX_00# run ping 192.168.12.1 interface ge-0/0/0 count 3
PING 192.168.12.1 (192.168.11.1): 56 data bytes
ping: sendto: No route to host
ping: sendto: No route to host
ping: sendto: No route to host

--- 192.168.12.1 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

No route! Oh no! *Do you have the correct routing and policy statement in place to allow the traffic?* No, so we need to add it.

Step 5: To add the correct routing and policy statement you need to allow traffic between the two zones on vSRX_00:

```
root@vSRX_00> configure
Entering configuration mode

[edit]
root@vSRX_00#
set routing-options static route 0.0.0.0/0 next-hop 172.16.10.2
set security policies from-zone green to-zone red policy vSRX_00_to_vSRX_01 match source-address any
set security policies from-zone green to-zone red policy vSRX_00_to_vSRX_01 match destination-
address any
set security policies from-zone green to-zone red policy vSRX_00_to_vSRX_01 match application any
set security policies from-zone green to-zone red policy vSRX_00_to_vSRX_01 then permit
```

Step 6: Enable vSRX_01 to receive and allow traffic from red_net towards blue_net:

```
root@vSRX_01> configure
Entering configuration mode

[edit]
root@vSRX_01#
set routing-options static route 0.0.0.0/0 next-hop 172.16.10.1
set security policies from-zone red to-zone blue policy vSRX_00_to_vSRX_01 match source-address any
set security policies from-zone red to-zone blue policy vSRX_00_to_vSRX_01 match destination-
address any
set security policies from-zone red to-zone blue policy vSRX_00_to_vSRX_01 match application any
set security policies from-zone red to-zone blue policy vSRX_00_to_vSRX_01 then permit
```

Step 7: Now, ping from vSRX_00 and confirm the session on vSRX_01:

```
root@vSRX_00# run ping 192.168.12.1 interface ge-0/0/0 count 3
PING 192.168.12.1 (192.168.11.1): 56 data bytes
64 bytes from 192.168.12.1: icmp_seq=0 ttl=64 time=1.964 ms
64 bytes from 192.168.12.1: icmp_seq=1 ttl=64 time=0.677 ms
64 bytes from 192.168.12.1: icmp_seq=2 ttl=64 time=0.819 ms

--- 192.168.11.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.677/1.153/1.964/0.576 ms

root@vSRX_01# run show security flow session protocol icmp | refresh 1
Session ID: 12, Policy name: vSRX_00_to_vSRX_01/6, Timeout: 4, Valid
  In: 192.168.10.1/0 --> 192.168.12.1/53790;icmp, Conn Tag: 0x0, If: ge-0/0/1.0, Pkts: 1, Bytes: 84,
  Out: 192.168.12.1/53790 --> 192.168.10.1/0;icmp, Conn Tag: 0x0, If: .local..0, Pkts: 1, Bytes: 84,
```

TIPS In a lab environment you can simulate a site-to-site VPN tunnel using this topology.

Building Your Second Topology

Guess what you are trying to achieve in this second topology? It's high availability (HA). Here's a quick checklist of what you need to accomplish this task:

- Two new vSRX VMs
- Two new virtual networks for control and fabric connections

For the vSRX high availability (chassis cluster) topology, you need a pair of the same version vSRX instances. A high availability pair can have only two members, sometimes also called a *cluster pair*. See Figure 3.2.

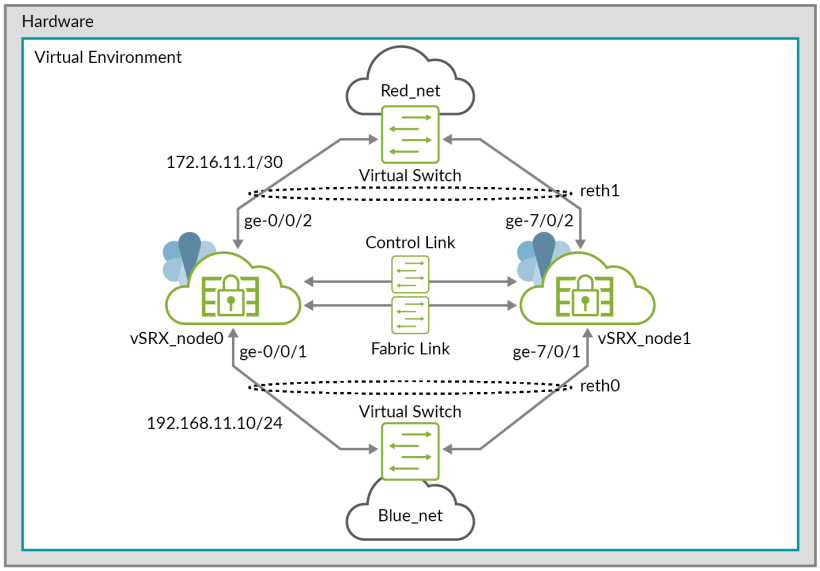


Figure 3.2 *Topology for vSRX Instances in High Availability*

HA SRX cluster has two unique links, namely, control and fabric links. The control link is used for cluster communication and for configuration synchronization. The fabric link on the other side is used for synchronizing the RTOs (real-time objects, that is sessions, etc.). You connect these links using isolated virtual networks.

Let's get started.

On the host, create two new isolated networks ctrl_net and fab_net for control link and fabric link, respectively.

Step 1: Create a file with name ctrl_net.xml at /etc/libvirt/qemu/networks and copy and paste the following snippet:

```
root@LabHost:~# nano /etc/libvirt/qemu/networks/ctrl_net.xml
<network>
  <name>ctrl_net</name>
  <bridge name='ctrl_net' stp='on' delay='0'/>
  <ip address='192.168.126.1' netmask='255.255.255.0'>
    <dhcp>
     <range start='192.168.126.100' end='192.168.126.250'/>
    </dhcp>
  </ip>
</network>
root@LabHost:~# more /etc/libvirt/qemu/networks/ctrl_net.xml
<network>
  <name>ctrl_net</name>
  <bridge name='ctrl_net' stp='on' delay='0'/>
```

```
  <ip address='192.168.126.1' netmask='255.255.255.0'>
    <dhcp>
     <range start='192.168.126.100' end='192.168.126.250'/>
    </dhcp>
  </ip>
</network>
```

Step 2: Define and start the network:

```
root@LabHost:~# virsh net-define /etc/libvirt/qemu/networks/ctrl_net.xml
Network ctrl_net defined from /etc/libvirt/qemu/networks/ctrl_net.xml
root@LabHost:~# virsh net-start ctrl_net
Network ctrl_net started
root@LabHost:~# virsh net-autostart ctrl_net
Network ctrl_net marked as autostarted
```

Step 3: On the same lines, create another network for fabric link as fab_net:

```
root@LabHost:~# nano /etc/libvirt/qemu/networks/fab_net.xml
root@LabHost:~# virsh net-define /etc/libvirt/qemu/networks/fab_net.xml
Network fab_net defined from /etc/libvirt/qemu/networks/fab_net.xml

root@LabHost:~# virsh net-start fab_net
Network fab_net started

root@LabHost:~# virsh net-autostart fab_net
Network fab_net marked as autostarted

root@LabHost:~# more /etc/libvirt/qemu/networks/fab_net.xml
<!--
WARNING: THIS IS AN AUTO-GENERATED FILE. CHANGES TO IT ARE LIKELY TO BE
OVERWRITTEN AND LOST. Changes to this xml configuration should be made using:
  virsh net-edit fab_net
or other application using the libvirt API.
-->

<network>
  <name>fab_net</name>
  <uuid>280cea77-6552-40d4-8caf-98aa42a2e578</uuid>
  <bridge name='fab_net' stp='on' delay='0'/>
  <mac address='52:54:00:17:ba:13'/>
  <ip address='192.168.127.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.127.100' end='192.168.127.250'/>
    </dhcp>
  </ip>
</network>
root@LabHost:~#
```

Step 4: Verify that you now have five networks on the host:

```
root@LabHost:~# virsh net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 blue_net             active     yes           yes
 ctrl_net             active     yes           yes
 default              active     yes           yes
 fab_net              active     yes           yes
 green_net            active     yes           yes
 red_net              active     yes           yes
```

Step 5: Now that the networks are ready, let's jump in to spin the two VMs that are Node0 and Node1 as primary and backup for a high availability chassis cluster. The first thing to do is to copy two more images from the master image. These are named imagev3_node0.qcow2 and imagev3_node1.qcow2:

```
root@LabHost:~# cp /var/lib/libvirt/images/junos-vsrx3-x86-64-18.4R1.8.qcow2 /var/lib/libvirt/
images/img_vSRX_node0.qcow2
root@LabHost:~# cp /var/lib/libvirt/images/junos-vsrx3-x86-64-18.4R1.8.qcow2 /var/lib/libvirt/
images/img_vSRX_node1.qcow2
root@LabHost:~# ls /var/lib/libvirt/images/
root@LabHost:~# ls -la /var/lib/libvirt/images/
total 3830932
drwx--x--x 2 root         root      4096 Jan 27 16:51 .
drwxr-xr-x 7 root         root      4096 Jan 26 06:36 ..
-rw-r--r-- 1 libvirt-qemu kvm  864878592 Jan 27 16:51 img_vSRX_00.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  861077504 Jan 27 16:52 img_vSRX_01.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 27 16:51 img_vSRX_node0.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 27 16:51 img_vSRX_node1.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 26 05:22 junos-vsrx3-x86-64-18.4R1.8.qcow2
```

Step 6: Install a VM named vSRX_node0:

```
root@LabHost:~# virt-install --name vSRX_
node0 --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/libvirt/images/
img_vSRX_node0.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:ctrl_
net,model=virtio --network=network:fab_net,model=virtio --network=network:blue_
net,model=virtio --network=network:red_net,model=virtio
root@LabHost:~# virt-install --name vSRX_
node0 --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/libvirt/images/
img_vSRX_node0.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:ctrl_
net,model=virtio --network=network:fab_net,model=virtio --network=network:blue_
net,model=virtio --network=network:red_net,model=virtio

Starting install...
Creating domain...     0 B  00:00:06

(virt-viewer:10220): GSpice-WARNING **: PulseAudio context failed Connection refused

(virt-viewer:10220): GSpice-WARNING **: pa_context_connect() failed: Connection refused
Domain creation completed.
```

Step 7: Follow Step 6 to create another VM, let's call it vSRX_node1. Two things need to be changed — the VM name and the image file name:

```
root@LabHost:~# virt-install --name vSRX_
node1 --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/libvirt/images/
img_vSRX_node1.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:ctrl_
net,model=virtio --network=network:fab_net,model=virtio --network=network:blue_
net,model=virtio --network=network:red_net,model=virtio

Starting install...
Creating domain...                                                                    |    0 B
00:00:06

(virt-viewer:10416): GSpice-WARNING **: PulseAudio context failed Connection refused
```

```
(virt-viewer:10416): GSpice-WARNING **: pa_context_connect() failed: Connection refused
Domain creation completed.
```

Step 8: Finally, the VMs are installed, let's check the status:

```
root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     vSRX_00                        running
 3     vSRX_01                        running
 4     vSRX_node0                     running
 5     vSRX_node1                     running
```

This brings us to the completion of the two requirements. Do you remember what they were?... Configuring isolated virtual networks and instantiating VMs.

Next, head into one VM at a time and configure them to take their tags, of master and backup. Let's use the following steps to configure the two vSRX VMs as an HA pair:

Step 1: On the vSRX console, log in to vSRX_00 to enable chassis cluster and assign it role of node0:

```
Command from Operational mode: - root> set chassis cluster cluster-id 1 node 0 reboot

root@LabHost:~# virsh console vSRX_node0
Connected to domain vSRX_node0
Escape character is ^]

FreeBSD/amd64 (Amnesiac) (ttyu0)
login: root
--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@:~ # cli
root> set chassis cluster cluster-id 1 node 0 reboot
Successfully enabled chassis cluster. Going to reboot now.

root>
*** FINAL System shutdown message from root@ ***
System going down IMMEDIATELY
```

Upon reboot, the solo root prompt would change as follows:

```
FreeBSD/amd64 (Amnesiac) (ttyu0)
login: root
Last login: Sun Jan 27 11:27:56 on ttyu0
--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@:~ #
root@:~ # cli
{primary:node0}
root>
```

Step 2: Similarly, console log in to vSRX_01 to enable chassis cluster and assign it the role of node1. Upon reboot, the node1 should take ownership as secondary:

```
root> set chassis cluster cluster-id 1 node 1 reboot
FreeBSD/amd64 (Amnesiac) (ttyu0)
login: root
Last login: Sun Jan 27 11:29:47 on ttyu0
```

```
--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@:~ #
root@:~ # cli
{secondary:node1}
root>
```

Step 3: Log back in to node0 to check the cluster status:

```
root> show chassis cluster status
{primary:node0}
root> show chassis cluster status
Monitor Failure codes:
    CS  Cold Sync monitoring        FL  Fabric Connection monitoring
    GR  GRES monitoring             HW  Hardware monitoring
    IF  Interface monitoring        IP  IP monitoring
    LB  Loopback monitoring         MB  Mbuf monitoring
    NH  Nexthop monitoring          NP  NPC monitoring
    SP  SPU monitoring              SM  Schedule monitoring
    CF  Config Sync monitoring      RE  Relinquish monitoring

Cluster ID: 1
Node   Priority Status               Preempt Manual   Monitor-failures

Redundancy group: 0 , Failover count: 1
node0  1        primary              no      no       None
node1  1        secondary            no      no       None
```

Step 4: Next, check the interfaces configured. You will be amazed to see node0 showing node1 interfaces, too:

```
root> show interface terse
```

During the `virt-install` command, we added the networks in the following order and the same interface mapping is displayed as shown in Table 3.2. In the following table "ge-7-x-x" denotes the interface on node1.

Table 3.2 Network to Interface Mapping

Network	Node0	Node1
default	fxp0.0	fxp0.0
ctrl_net	em0.0	em0.0
fab_net	ge-0/0/0	ge-7/0/0
blue_net	ge-0/0/1	ge-7/0/1
red_net	ge-0/0/2	ge-7/0/2

NOTE The fab interface defined here is ge-0/0/0, in the real world it can be any interface that a user wishes to assign to it.

The vSRX cluster uses two interfaces exclusively for clustering:

- Cluster control link (em0)

- Cluster fabric links (fab0 and fab1). For example, you can specify ge-0/0/0 as fab0 on node0 and ge-7/0/0 as fab1 on node1:

```
{primary:node0}
root> show interfaces terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
gr-0/0/0                up    up
ip-0/0/0                up    up
lt-0/0/0                up    up
ge-0/0/1                up    up
ge-0/0/2                up    up
ge-7/0/0                up    up
ge-7/0/1                up    up
ge-7/0/2                up    up
dsc                     up    up
em0                     up    up
em0.0                   up    up   inet     129.16.0.1/2
                                            143.16.0.1/2
                                    tnp      0x1100001
fab0                    up    down
fab0.0                  up    down inet     30.17.0.200/24
fab1                    up    down
fab1.0                  up    down inet     30.18.0.200/24
fti0                    up    up
fxp0                    up    up
fxp0.0                  up    up
```

Step 5: Set the root password:

```
root@vSRX_Node0# set system root-authentication plain-text-password
New password:
Retype new password:
```

Step 6: Finally, let's configure the cluster with fab interface and green and red interfaces:

```
set apply-groups "${node}"
set groups node0 system host-name vSRX_Node0
set groups node1 system host-name vSRX_Node1

set interfaces fab0 fabric-options member-interfaces ge-0/0/0
set interfaces fab1 fabric-options member-interfaces ge-7/0/0

set chassis cluster redundancy-group 0 node 0 priority 100
set chassis cluster redundancy-group 0 node 1 priority 1
set chassis cluster redundancy-group 1 node 0 priority 100
set chassis cluster redundancy-group 1 node 1 priority 1

set chassis cluster reth-count 2

set interfaces ge-0/0/1 gigether-options redundant-parent reth0
set interfaces ge-7/0/1 gigether-options redundant-parent reth0
set interfaces reth0 redundant-ether-options redundancy-group 1
set interfaces reth0 unit 0 family inet address 192.168.11.10/24
set security zones security-zone blue host-inbound-traffic system-services all
set security zones security-zone blue host-inbound-traffic protocols all
set security zones security-zone blue interfaces reth0.0
```

```
set interfaces ge-0/0/2 gigether-options redundant-parent reth1
set interfaces ge-7/0/2 gigether-options redundant-parent reth1
set interfaces reth1 redundant-ether-options redundancy-group 1
set interfaces reth1 unit 0 family inet address 172.16.11.1/30
set security zones security-zone red host-inbound-traffic system-services all
set security zones security-zone red host-inbound-traffic protocols all
set security zones security-zone red interfaces reth1.0
```

Notice that once you commit the configuration, the commit is applied to both the nodes:

```
{primary:node0}[edit]
root# commit
node0:
configuration check succeeds
node1:
commit complete
node0:
commit complete
```

Step 7: Check the cluster interface status using the following command.

The fab interface should be `Up/Up` and the reth interface status should be `Up`:

```
root@vSRX_Node0# run show chassis cluster interfaces
{primary:node0}[edit]
root@vSRX_Node0# run show chassis cluster interfaces
Control link status: Up

Control interfaces:
    Index   Interface   Monitored-Status   Internal-SA   Security
    0       em0         Up                 Disabled      Disabled

Fabric link status: Up

Fabric interfaces:
    Name    Child-interface    Status                    Security
                               (Physical/Monitored)
    fab0    ge-0/0/0           Up   / Up                 Disabled
    fab0
    fab1    ge-7/0/0           Up   / Up                 Disabled
    fab1

Redundant-ethernet Information:
    Name         Status      Redundancy-group
    reth0        Up          1
    reth1        Up          1
Redundant-pseudo-interface Information:
    Name         Status      Redundancy-group
    lo0          Up          0
```

Step 8: Finally, check the status of the cluster. The status must show 'Up' for both RGs (RG0 and RG1):

```
{primary:node0}[edit]
root@vSRX_Node0# run show chassis cluster status
Monitor Failure codes:
    CS  Cold Sync monitoring        FL  Fabric Connection monitoring
    GR  GRES monitoring             HW  Hardware monitoring
    IF  Interface monitoring        IP  IP monitoring
```

```
    LB  Loopback monitoring          MB  Mbuf monitoring
    NH  Nexthop monitoring           NP  NPC monitoring
    SP  SPU monitoring               SM  Schedule monitoring
    CF  Config Sync monitoring       RE  Relinquish monitoring

Cluster ID: 1
Node   Priority Status               Preempt Manual   Monitor-failures
Redundancy group: 0 , Failover count: 1
node0  100      primary              no      no       None
node1  1        secondary            no      no       None
Redundancy group: 1 , Failover count: 1
node0  100      primary              no      no       None
node1  1        secondary            no      no       None
```

The topology is now complete with the two vSRX VMs configured in a high availability pair.

Building Your Third Topology

First, let's visualize this next exercise in Figure 3.3.

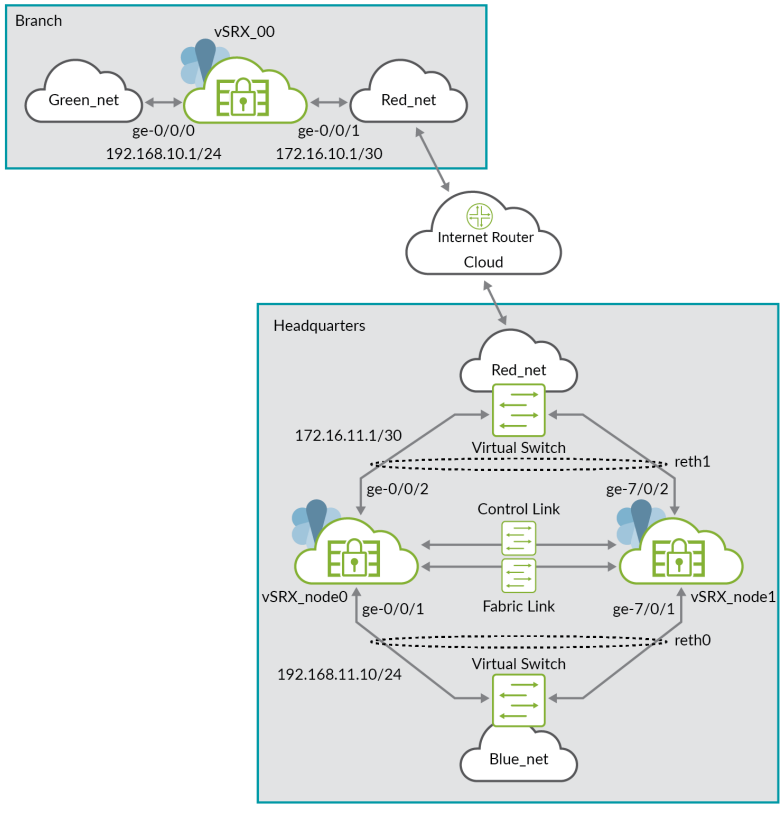


Figure 3.3 Topology for vSRX Instances in Headquarter to Branch Setup

Doesn't Figure 3.3 look familiar? Yes, it is! We built one part of this topology in Topology 1 of this chapter, and the second part was created in Topology 2. Now, in Topology 3, let's use the first two topologies and simulate a production setup of a branch office connecting to its headquarters location. Branch location setup is connected to the green_net on its one side via interface ge-0/0/0 and red_net on the other side to the interface ge-0/0/1. Likewise, HQ cluster is connected to the blue_net on reth0 and red_net on reth1.

The red_net from each side of the branch and HQ is connected to another VM acting as an Internet router for this topology. This acts as a routing device between the branch and HQ location.

So, let's get going and prep Topology 3. First a list: what do we already have and what do we need next?

- A high availability (chassis cluster) setup in HQ – Configured in Topology 2
- VM instance in branch node – Configured in Topology 1 [let's use vSRX_00]
- Router as Internet cloud – We need to create this VM

Before configuring the HQ and branch vSRX devices you need to create another VM that can act as an Internet router. Let's first create another vSRX VM, then configure it to act as an Internet router.

Then let's use the following steps to configure an Internet router named vSRX_Int_Router.

Step 1: Copy another image from the master vSRX image with name image3_int_router.qcow2:

```
root@LabHost:~# cp /var/lib/libvirt/images/junos-vsrx3-x86-64-18.4R1.8.qcow2 /var/lib/libvirt/
images/img_Internet_Router.qcow2
root@LabHost:~# ls -la /var/lib/libvirt/images/
total 4824156
drwx--x--x 2 root         root      4096 Jan 28 08:47 .
drwxr-xr-x 7 root         root      4096 Jan 26 06:36 ..
-rw-r--r-- 1 root         root 732299264 Jan 28 08:47 img_Internet_Router.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  866385920 Jan 28 08:47 img_vSRX_00.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  862781440 Jan 28 08:47 img_vSRX_01.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  876085248 Jan 28 08:47 img_vSRX_node0.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  870055936 Jan 28 08:47 img_vSRX_node1.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 26 05:22 junos-vsrx3-x86-64-18.4R1.8.qcow2
```

Step 2: Using the `virt-install` command, instantiate the VM:

```
virt-install --name vSRX_Internet_
Router --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/libvirt/images/
img_Internet_Router.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:red_
net,model=virtio --network=network:red_net,model=virtio
root@LabHost:~# virt-install --name vSRX_Internet_
Router --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/libvirt/images/
img_Internet_Router.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --network=network:red_
```

```
net,model=virtio --network=network:red_net,model=virtio

Starting install...
Creating domain...        |    0 B  00:00:04

(virt-viewer:18616): GSpice-WARNING **: PulseAudio context failed Connection refused

(virt-viewer:18616): GSpice-WARNING **: pa_context_connect() failed: Connection refused
Domain creation completed.
```

Step 3: Log in to the VM using the console, and configure the root password and hostname:

```
root# set system host-name Internet_Router
root# set system root-authentication plain-text-password
New password:
Retype new password:
root@LabHost:~# virsh console vSRX_Internet_Router
Connected to domain vSRX_Internet_Router
Escape character is ^]


FreeBSD/amd64 (Amnesiac) (ttyu0)
login: root

--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@:~ # cli
root> configure
Entering configuration mode
[edit]
root# set system host-name Internet_Router
root# set system root-authentication plain-text-password
New password:
Retype new password:

[edit]
root# commit
commit complete
```

Step 4: Configure the vSRX VM to act as a router, use the following commands, and commit:

NOTE Remember, when creating the VM for the Internet router, we added three networks to the VM ---namely default, red_net, and red_net. This means that the ge-0/0/0 and ge-0/0/1 on the new VM are both part of red_net. Both interfaces have separate subnets with one side connecting to Branch and the other to HQ:

```
set security zones security-zone red interfaces ge-0/0/0.0
set security zones security-zone red interfaces ge-0/0/1.0
set interfaces ge-0/0/0 unit 0 family inet address 172.16.10.2/30
set interfaces ge-0/0/1 unit 0 family inet address 172.16.11.2/30
set routing-options static route 192.168.10.0/24 next-hop 172.16.10.1
set routing-options static route 192.168.11.0/24 next-hop 172.16.11.1
```

Step 5: Confirm that you are able to ping to the Branch and the HQ red network side interface IP addresses:

```
[edit]
root@Internet_Router# run ping 172.16.10.1 count 2
PING 172.16.10.1 (172.16.10.1): 56 data bytes
64 bytes from 172.16.10.1: icmp_seq=0 ttl=64 time=1.807 ms
64 bytes from 172.16.10.1: icmp_seq=1 ttl=64 time=0.748 ms
--- 172.16.10.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.748/1.277/1.807/0.530 ms

[edit]
root@Internet_Router# run ping 172.16.11.1 count 2
PING 172.16.11.1 (172.16.11.1): 56 data bytes
64 bytes from 172.16.11.1: icmp_seq=0 ttl=64 time=36.931 ms
64 bytes from 172.16.11.1: icmp_seq=1 ttl=64 time=0.785 ms
--- 172.16.11.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.785/18.858/36.931/18.073 ms
```

Step 6: Configure the Branch VM, IR, and HQ cluster to allow traffic pass from vSRX_00 green_net side to HQ blue_net side.

Now that the topology is built, it's time to configure the branch and HQ cluster to allow traffic between green_net on branch side to reach green_net on HQ side.

Branch Side Configuration:

```
set security address-book global address green_net 192.168.10.0/24
set security address-book global address hq_blue_net 192.168.11.0/24
set security policies from-zone green to-zone red policy branch_to_HQ match source-address green_net
set security policies from-zone green to-zone red policy branch_to_HQ match destination-address hq_
blue_net
set security policies from-zone green to-zone red policy branch_to_HQ match application any
set security policies from-zone green to-zone red policy branch_to_HQ then permit
set security address-book global address green_net 192.168.10.0/24
set security address-book global address hq_green_net 192.168.11.0/24
set routing-options static route 0.0.0.0/0 next-hop 172.16.10.2
```

IR Configuration:

```
set security policies default-policy permit-all
```

HQ Configuration:

```
set security address-book global address green_net 192.168.10.0/24
set security address-book global address hq_blue_net 192.168.11.0/24
set security policies from-zone red to-zone blue policy branch_to_HQ match source-address green_net
set security policies from-zone red to-zone blue policy branch_to_HQ match destination-address hq_
blue_net
set security policies from-zone red to-zone blue policy branch_to_HQ match application any
set security policies from-zone red to-zone blue policy branch_to_HQ then permit
set routing-options static route 0.0.0.0/0 next-hop 172.16.11.2
```

Step 7: Confirm that the VM in branch can ping to HQ side green network interface:

```
[edit]
root@vSRX_00# run ping 192.168.11.10 interface ge-0/0/0 count 2
PING 192.168.11.10 (192.168.11.10): 56 data bytes
```

```
64 bytes from 192.168.11.10: icmp_seq=0 ttl=63 time=6.603 ms
64 bytes from 192.168.11.10: icmp_seq=1 ttl=63 time=1.335 ms

--- 192.168.11.10 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.335/3.969/6.603/2.634 ms
root@vSRX_Node0> show security flow session protocol icmp | refresh 2
---(refreshed at 2019-01-28 04:26:09 UTC)---
node0:
--------------------------------------------------------------------------
Session ID: 24, Policy name: branch_to_HQ/6, State: Active, Timeout: 2, Valid
  In: 192.168.10.1/0 --> 192.168.11.10/64291;icmp, Conn Tag: 0x0, If: reth1.0, Pkts: 1, Bytes: 84,
  Out: 192.168.11.10/64291 --> 192.168.10.1/0;icmp, Conn Tag: 0x0, If: .
local..0, Pkts: 1, Bytes: 84,

Session ID: 26, Policy name: branch_to_HQ/6, State: Active, Timeout: 4, Valid
  In: 192.168.10.1/1 --> 192.168.11.10/64291;icmp, Conn Tag: 0x0, If: reth1.0, Pkts: 1, Bytes: 84,
  Out: 192.168.11.10/64291 --> 192.168.10.1/1;icmp, Conn Tag: 0x0, If: .
local..0, Pkts: 1, Bytes: 84,
Total sessions: 2
```

Lab Challenge 1: Create one more VM as a host behind each vSRX VM green_net/blue_net and ping between the two hosts.

Building Your Fourth Topology

First, let's visualize this next exercise shown in Figure 3.4.

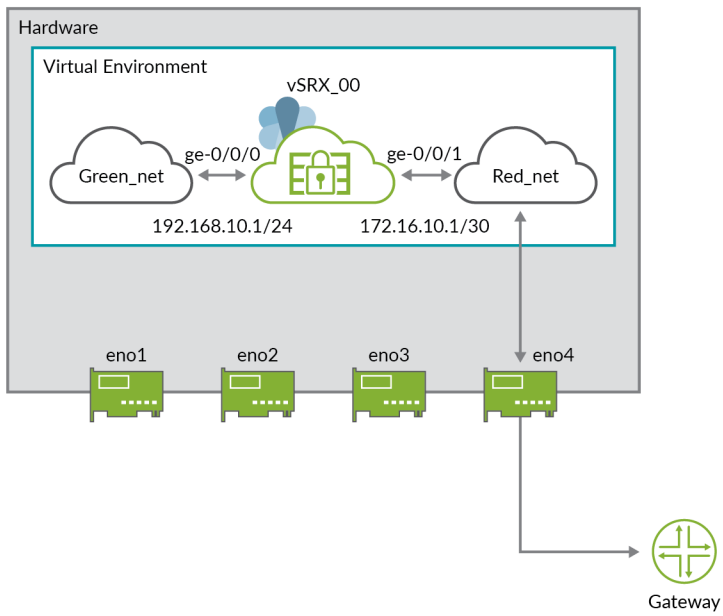


Figure 3.4 *Topology for Adding Interface to Virtual Network*

In the last three topologies, we did a hands-on exercise to create VMs and join the VMs to virtual networks. In this lab exercise, you'll reuse the first VM (vSRX_00) that you created and connect that VM to the host physical NIC to a gateway (a Layer 3 hop external to the host). In this Fourth Topology, we'll learn the new concept of how to add an interface to an existing virtual network.

Here is a quick checklist of what's necessary to build the topology:

- vSRX VM
- Connecting the VMs VN to pNIC
- Physical NIC available on host
- Cable connecting from pNIC to gateway
- Gateway configured with a reachable IP address

To check and configure the VM, use the following steps.

Step 1: Check if the vSRX_00V M is running:

```
root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 2     vSRX_00                        running
 4     vSRX_node0                     running
 5     vSRX_node1                     running
 7     vSRX_Internet_Router           running
 8     vSRX_01                        running
```

NOTE Shut off VM vSRX_01 from Topology 1, as it is already part of red_net and is configured with the other /30 IP address of the subnet. In this topology, we used the IP on the gateway. Use the "virsh destroy vSRX_01" command to shut off the VM.

Step 2: Check the virtual networks associated with the said VM:

```
root@LabHost:~# virsh domiflist vSRX_00
Interface  Type       Source     Model       MAC
-------------------------------------------------------------
vnet0      network    default    virtio      52:54:00:86:82:c2
vnet1      network    green_net  virtio      52:54:00:25:e1:06
vnet2      network    red_net    virtio      52:54:00:9f:5e:6e
```

Step 3: Check the details of network red_net with bridge_utils command `brctl`:

```
root@LabHost:~# brctl show red_net
bridge name     bridge id               STP enabled     interfaces
red_net         8000.525400162dc5       yes             red_net-nic
                                                        vnet10
                                                        vnet15
                                                        vnet17
                                                        vnet18
                                                        vnet2
```

Step 4: Add the required physical interface to the bridge by using the bridge-utility `brctl`:

```
root@LabHost:~# brctl addif red_net eno4
root@LabHost:~# brctl show red_net
bridge name     bridge id               STP enabled     interfaces
red_net         8000.246e96a97155       yes             eno4
                                                        red_net-nic
                                                        vnet10
                                                        vnet15
                                                        vnet17
                                                        vnet18
                                                        vnet2
```

NOTE Note that eno4 is the physical interface in the lab host server connected to the gateway router.

Step 5: Confirm that the physical link is connected. Use the following commands to check:

```
root@LabHost:~# ip link
3: eno4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master red_
net state UP mode DEFAULT group default qlen 1000
    link/ether 24:6e:96:a9:71:55 brd ff:ff:ff:ff:ff:ff
```

TIPS `UP` state means that the interface is connected with the cable. `NO_CARRIER` confirms that no cable connected to the port.

Step 6: Check the connectivity. Connect to the vSRX VMs and ping to the next hop device:

```
root@vSRX_00> show interfaces terse | match “ge-0/0”
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     192.168.10.1/24
ge-0/0/1                up    up
ge-0/0/1.0              up    up   inet     172.16.10.1/30

root@vSRX_00> ping 172.16.10.2 count 2
PING 172.16.10.2 (172.16.10.2): 56 data bytes
64 bytes from 172.16.10.2: icmp_seq=0 ttl=64 time=17.451 ms
64 bytes from 172.16.10.2: icmp_seq=1 ttl=64 time=9.403 ms
--- 172.16.10.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 9.403/13.427/17.451/4.024 ms
```

Step 6: Confirm that the packets are actually being sent to the physical interface. Let’s use the tcpdump utility on vnet2 and eno4:

```
root@LabHost:~# tcpdump -nni vnet2 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on vnet2, link-type EN10MB (Ethernet), capture size 262144 bytes
11:12:46.331581 IP 172.16.10.1 > 172.16.10.2: ICMP echo request, id 22308, seq 0, length 64
11:12:46.345728 IP 172.16.10.2 > 172.16.10.1: ICMP echo reply, id 22308, seq 0, length 64
11:12:47.329925 IP 172.16.10.1 > 172.16.10.2: ICMP echo request, id 22308, seq 1, length 64
11:12:47.338636 IP 172.16.10.2 > 172.16.10.1: ICMP echo reply, id 22308, seq 1, length 64
```

```
root@LabHost:~# tcpdump -nni eno4 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eno4, link-type EN10MB (Ethernet), capture size 262144 bytes
11:12:46.331598 IP 172.16.10.1 > 172.16.10.2: ICMP echo request, id 22308, seq 0, length 64
11:12:46.345705 IP 172.16.10.2 > 172.16.10.1: ICMP echo reply, id 22308, seq 0, length 64
11:12:47.329942 IP 172.16.10.1 > 172.16.10.2: ICMP echo request, id 22308, seq 1, length 64
11:12:47.338631 IP 172.16.10.2 > 172.16.10.1: ICMP echo reply, id 22308, seq 1, length 64
```

Building Your Fifth Topology

In the Fourth Topology, you connected the VM via a virtual network to a physical NIC. In the Fifth Topology, you will learn to bypass connecting the VM to any of the virtual networks and connect to the physical NIC directly, as shown in Figure 3.5.

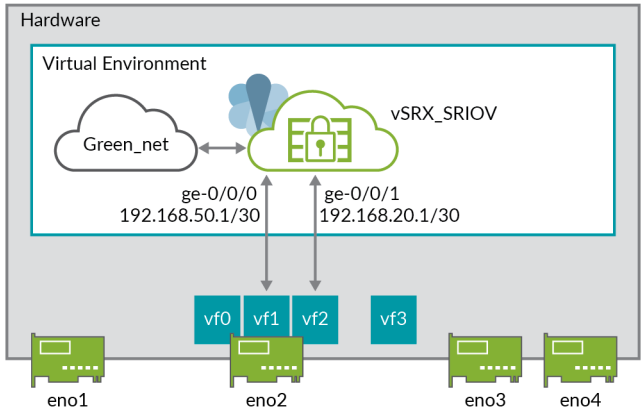


Figure 3.5 *Topology for Connecting VM to Physical NIC Using SRIOV*

The technology here is called SR-IOV (single-root I/O virtualization). SR-IOV extends the concept of virtualized functions to pNIC. The single physical NIC card can be divided into up to 16 partitions. NIC maintains different queues. Each of the queues can be plugged into VMs directly, as separate interfaces, bypassing the hypervisor completely.

MORE? For more information about SR-IOV, see: https://www.juniper.net/documentation/en_US/junos/topics/concept/disaggregated-junos-sr-iov.html.

Juniper supports Intel 82599, X520/540, X710/XL710, Mellanox ConnectX-3, and ConnectX-4 Family adapters.

MORE? And for more details about what Juniper supports, see: https://www.juniper.net/documentation/en_US/vsrx/topics/task/configuration/security-vsrx-kvm-add-sr-iov-interfaces.html.

Here is a checklist of what's necessary to complete the Fifth Topology:

- Prepare the host to allow using SR-IOV
- Confirm the virtual function PCI address to be used
- Spin a VM with only a default virtual network
- Confirm connectivity

Jump into your host and use the following steps to prepare it.

Step 1: Insert and confirm that a Juniper supported SR-IOV NIC is inserted into the host:

```
root@LabHost:~# lshw -c network -businfo
PCI (sysfs)     ----  (It does take some time here to retrieve the output)

Bus info          Device         Class          Description
=========================================================
pci@0000:01:00.0  eno1           network        82599ES 10-Gigabit SFI/SFP+ Network Connection
pci@0000:01:00.1  eno2           network        82599ES 10-Gigabit SFI/SFP+ Network Connection
pci@0000:07:00.0  eno3           network        I350 Gigabit Network Connection
pci@0000:07:00.1  eno4           network        I350 Gigabit Network Connection
```

Here, Intel 82599 is in slots 1 and 2, and we'll use eno2 for SR-IOV connections.

Step 2: Enable the Intel VT-d CPU virtualization extensions in BIOS. Connect to the console of your host server and navigate to the BIOS setting. On this server, navigate to Virtualization Technology under Launch System Setup > System BIOS > System BIOS Settings > Processor Setting (see Figure 3.6).

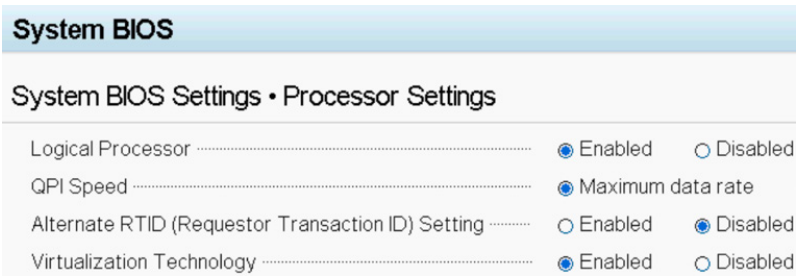


Figure 3.6 *System BIOS Settings for SR-IOV (Processor)*

NOTE Verify the process with the vendor because different systems have different methods to enable VT-d.

Step 3: Further, SR-IOV global is required to be set to Enabled in BIOS.

On this server, SR-IOV Global Enable is located by navigating System BIOS > System BIOS Settings > Integrated Devices and set it to Enabled, as shown in Figure 3.7.

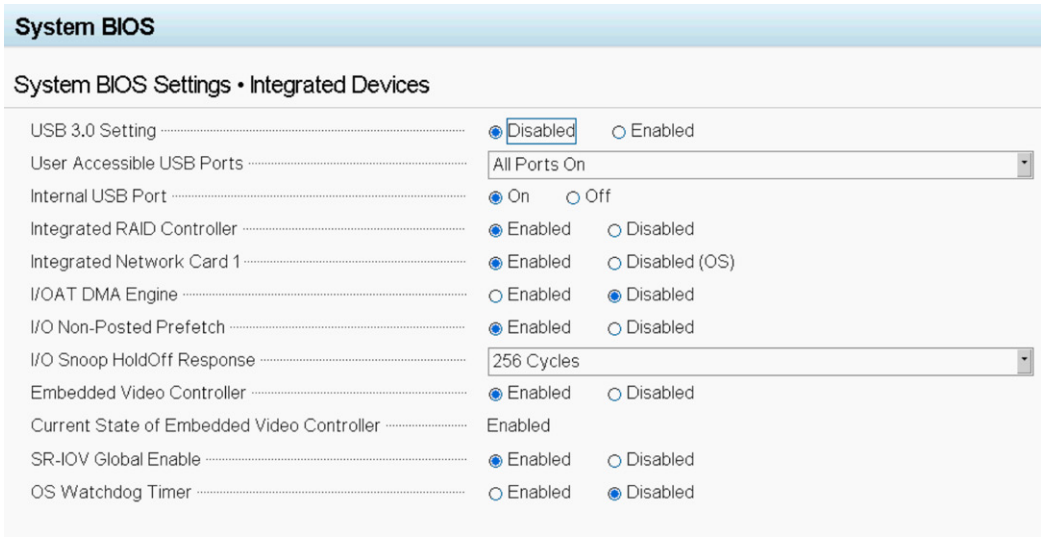


Figure 3.7 System BIOS Settings for SR-IOV (Integrated Devices)

Step 4: Next, we need to enable IOMMU (input-output memory management unit). This is required as SR-IOV virtual functions are queues connecting directly to the VM.

```
root@LabHost:~# echo intel_iommu=on > /boot/grub/grub.conf
```

Step 5: Update Grub and reboot the system:

```
root@LabHost:~# GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on"
root@LabHost:~# update-grub

root@LabHost:~# update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.16.0-30-generic
Found initrd image: /boot/initrd.img-3.16.0-30-generic
Found memtest86+ image: /memtest86+.elf
Found memtest86+ image: /memtest86+.bin
done
```

Step 6: Reboot the host using the keyword "reboot".

Now that you have enabled the host with the capability to utilize SR-IOV functionality, follow the steps below to divide the SR-IOV NIC, eno2, into different virtual functions and then spin up the VM.

Step 1: Define four virtual functions for eno2 interface, update the sriov_numvfs file with number 4:

```
root@LabHost:~# echo 4 > /sys/class/net/eno2/device/sriov_numvfs
root@LabHost:~# more /sys/class/net/eno2/device/sriov_numvfs
```

Step 2: Recheck the hardware details to see that the virtual functions are listed correctly:

```
root@LabHost:~# lshw -c network -businfo
Bus info          Device         Class          Description
========================================================
pci@0000:01:00.0  eno1           network        82599ES 10-Gigabit SFI/SFP+ Network Connection
pci@0000:01:00.1  eno2           network        82599ES 10-Gigabit SFI/SFP+ Network Connection
pci@0000:01:10.1  eth0           network        Illegal Vendor ID
pci@0000:01:10.3  eth1           network        Illegal Vendor ID
pci@0000:01:10.5  eth2           network        Illegal Vendor ID
pci@0000:01:10.7  eth3           network        Illegal Vendor ID
pci@0000:07:00.0  eno3           network        I350 Gigabit Network Connection
pci@0000:07:00.1  eno4           network        I350 Gigabit Network Connection
```

You can clearly see the difference: eno2 now has four new interfaces, that is, virtual functions specifically denoted as eth0, eth1, eth2, and eth3.

Step 3: Create an image copy from the master vSRX file:

```
root@LabHost:~# cp /var/lib/libvirt/images/junos-vsrx3-x86-64-18.4R1.8.qcow2 /var/lib/libvirt/
images/img_vSRX_SRIOV.qcow2
root@LabHost:~# ls -la /var/lib/libvirt/images/
total 5671648
drwx--x--x 2 root         root      4096 Jan 29 16:01 .
drwxr-xr-x 7 root         root      4096 Jan 29 15:31 ..
-rw-r--r-- 1 libvirt-qemu kvm  864288768 Jan 28 11:27 img_Internet_Router.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  866844672 Jan 28 11:27 img_vSRX_00.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  864288768 Jan 28 11:00 img_vSRX_01.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  876937216 Jan 28 11:27 img_vSRX_node0.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  870776832 Jan 28 11:26 img_vSRX_node1.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 29 16:01 img_vSRX_SRIOV.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 26 05:22 junos-vsrx3-x86-64-18.4R1.8.qcow2
```

Step 4: Copy and paste the following command to instantiate the VM:

```
virt-install --name vSRX_SRIOV --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/
var/lib/libvirt/images/img_vSRX_SRIOV.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-
type linux --os-variant rhel7 --import --network=network:default,model=virtio --host-
device=pci_0000_01_10_1 --host-device=pci_0000_01_10_5

root@LabHost:~# virt-install --name vSRX_
SRIOV --ram 4096 --cpu SandyBridge, --vcpus=2 --arch=x86_64 --disk path=/var/lib/libvirt/images/
img_vSRX_SRIOV.qcow2,size=16,device=disk,bus=ide,format=qcow2 --os-type linux --os-
variant rhel7 --import --network=network:default,model=virtio --host-device=pci_0000_01_10_1 --host-
device=pci_0000_01_10_5
```

```
Starting install...
Creating domain...                          |    0 B  00:00:04
(virt-viewer:3187): GSpice-WARNING **: PulseAudio context failed Connection refused
(virt-viewer:3187): GSpice-WARNING **: pa_context_connect() failed: Connection refused
Domain creation completed.
```

Step 5: Check that the vSRX VM is running:

```
root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 1     vSRX_SRIOV                     running
 -     vSRX_00                        shut off
 -     vSRX_01                        shut off
 -     vSRX_Internet_Router           shut off
 -     vSRX_node0                     shut off
 -     vSRX_node1                     shut off
```

Step 6: Log in to the console using the `virsh` command and check `show interface` upon login:

```
root@LabHost:~# virsh console vSRX_SRIOV
Connected to domain vSRX_SRIOV
Escape character is ^]

FreeBSD/amd64 (Amnesiac) (ttyu0)
login: root
Last login: Thu Jan 17 11:12:30 on ttyu0

--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@:~ # cli
root>

root> show chassis fpc pic-status
Slot 0   Online       FPC
  PIC 0  Online       VSRX DPDK GE
root> show interfaces terse
Interface               Admin Link Proto    Local     Remote
ge-0/0/0                up    up
gr-0/0/0                up    up
ip-0/0/0                up    up
lsq-0/0/0               up    up
lt-0/0/0                up    up
mt-0/0/0                up    up
sp-0/0/0                up    up
sp-0/0/0.0              up    up   inet
                                   inet6
sp-0/0/0.16383          up    up   inet
ge-0/0/1                up    up
dsc                     up    up
```

Now that the VM is up and the SR-IOV interfaces are detected, let's configure the interfaces and then check the connectivity. What are we trying to achieve? We are bypassing the hypervisor and have connected the VM directly to the physical NIC by using virtual functions.

As we mentioned earlier, now the virtual functions are like queues that pass traffic to the NIC and then outside. So, in order to segregate traffic on the virtual functions (0 – 3), let's configure these VMs to explicity define VLANs on their interfaces. To summarize, multiple virtual functions will have multiple VLAN traffic on the pNIC, and the connected next hop switch should be a truck port to accept this traffic.

To configure and confirm the connectivity, use the following steps.

Step 1: Configure ge-0/0/0 and ge-0/0/1 with a VLAN and an IP address in the same subnet as its respective gateway address:

```
set security zones security-zone green host-inbound-traffic system-services all
set security zones security-zone green host-inbound-traffic protocols all
set security zones security-zone green interfaces ge-0/0/0.301
set interfaces ge-0/0/0 vlan-tagging
set interfaces ge-0/0/0 unit 301 vlan-id 301
set interfaces ge-0/0/0 unit 301 family inet address 192.168.50.1/30

set security zones security-zone red host-inbound-traffic system-services all
set security zones security-zone red host-inbound-traffic protocols all
set security zones security-zone red interfaces ge-0/0/1.302
set interfaces ge-0/0/1 vlan-tagging
set interfaces ge-0/0/1 unit 302 vlan-id 302
set interfaces ge-0/0/1 unit 302 family inet address 172.16.20.1/30
```

Step 2: Enable the next-hop gateway to accept tagged traffic from two different VLANs on the same interface. We used a Juniper EX Series switch configured in the following fashion.

You need to check your gateway type and configure it accordingly:

```
set interfaces ge-0/0/16 unit 0 family ethernet-switching vlan members vlan301
set interfaces irb unit 301 family inet address 192.168.50.2/30
set routing-instances DAYONE_GREEN instance-type virtual-router
set routing-instances DAYONE_GREEN interface irb.301
set vlans vlan301 vlan-id 301
set vlans vlan301 l3-interface irb.301

set interfaces ge-0/0/16 unit 0 family ethernet-switching vlan members vlan302
set interfaces irb unit 302 family inet address 172.16.10.2/30
set routing-instances DAYONE_RED instance-type virtual-router
set routing-instances DAYONE_RED interface irb.302
set vlans vlan302 vlan-id 302
set vlans vlan302 l3-interface irb.302
```

Step 3: Check the connectivity by using the ping utility:

```
[edit]
root# run ping 192.168.50.2 count 3
PING 192.168.50.2 (192.168.50.2): 56 data bytes
64 bytes from 192.168.50.2: icmp_seq=0 ttl=64 time=8.279 ms
64 bytes from 192.168.50.2: icmp_seq=1 ttl=64 time=7.810 ms
64 bytes from 192.168.50.2: icmp_seq=2 ttl=64 time=10.737 ms
```

```
--- 192.168.50.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 7.810/8.942/10.737/1.284 ms

[edit]
root# run ping 172.16.20.2 count 3
PING 172.16.10.2 (172.16.10.2): 56 data bytes
64 bytes from 172.16.10.2: icmp_seq=0 ttl=64 time=6.181 ms
64 bytes from 172.16.10.2: icmp_seq=1 ttl=64 time=5.652 ms
64 bytes from 172.16.10.2: icmp_seq=2 ttl=64 time=10.657 ms

--- 172.16.10.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 5.652/7.497/10.657/2.245 ms
```

Step 4: Confirm that the ping response is from the gateway; confirm the ARP learned on either side:

```
root@Contrail-DataSwitch# run show arp | match "ge-0/0/1|ge-0/0/0"
02:09:c0:28:1f:8e 172.16.20.1     172.16.20.1               irb.302 [ge-0/0/16.0]   none
02:09:c0:bf:3b:dd 192.168.50.1    192.168.50.1              irb.301 [ge-0/0/16.0]   none

root@DataSwitch# run show arp | match "irb.301|irb.302"
02:09:c0:28:1f:8e 172.16.20.1     172.16.20.1               irb.302 [ge-0/0/16.0]   none
02:09:c0:bf:3b:dd 192.168.50.1    192.168.50.1              irb.301 [ge-0/0/16.0]   none
```

Step 5: Check yourself with the "show interface" command to ensure that the MAC address is correct:

```
root@vSRX_SRIOV> show interfaces ge-0/0/0 | match Current
  Current address: 02:09:c0:bf:3b:dd, Hardware address: 02:09:c0:bf:3b:dd
root@vSRX_SRIOV> show interfaces ge-0/0/1 | match Current
  Current address: 02:09:c0:28:1f:8e, Hardware address: 02:09:c0:28:1f:8e
```

Well, that's all for this topology.

Check this yourself: the same MAC address will be defined in the "ip link" output for the two virtual functions assigned:

```
5: eno2: <BROADCAST,MULTICAST,UP,LOWER_
UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qlen 1000
    link/ether 24:6e:96:a9:71:52 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 02:09:c0:bf:3b:dd, spoof checking on, link-state auto, trust off
    vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto, trust off
    vf 2 MAC 02:09:c0:28:1f:8e, spoof checking on, link-state auto, trust off
    vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto, trust off
```

Summary

In this chapter you discovered how simple it is to deploy and interconnect multiple vSRXs on the same Linux host. You started by connecting two vSRX VMs over a red network, simulating what one can call a site-to-site topology. You next configured two vSRX VMs in high availability (chassis cluster) pair which can be used in a HQ location. You next merged the First Topology and the Second Topology together with another VM acting as an internet router, which one can call a branch to HQ topology. To conclude, you learned ways to connect an vSRX instance to the physical NIC of the host, initially using Linux bridges, and then bypassing the same for high performance requirements using SR-IOV.

Chapter 4

Troubleshooting vSRX on KVM

This chapter begins by familiarizing you with different utilities used to verify the host and the VM state. Next, it covers how to check the vSRX VM's Routing Engine and Packet Forwarding Engine state, and further in you'll learn how to confirm traffic flows over the virtual network. The chapter concludes by reviewing different log files on the host and the vSRX VM and how to use them.

Verify Host and VM State

To verify the VM's states and the networks configured and connected, you can generally use the following two Linux utilities:

- Virsh
- Brctl

More on Virsh (Virtual Shell) Command.

Recall that in Section 1 of Chapter 2, you installed some packages, and libvirt was one of them. Libvirt is an open source software for managing VMs. There is an API library, a daemon (libvirtd), and a command line utility (virsh). Juniper uses libvirt to create and manage vSRX instances.

Virsh, a command line user interface tool you have used extensively in this *Day One* book is used to create, start, pause, and shut down a domain (by domain we mean a VM). Usage:

```
virsh [OPTION]… <command> <domain> [ARG]…
```

- Where `OPTION` includes usage of '–v' for version check, `command` is like '`list`' which lists all VMs.
- `domain` is the numeric domain ID, or the domain name, or the domain uuid<id,name,uuid>.
- **`ARG`** are command-specific options.

For reference always use the "`--help`" option: -

```
virsh --help
```

Let's learn more about using the `virsh` command:

Step1: List all the VMs on your host:

```
root@LabHost:~# virsh list --all
 Id    Name                           State
----------------------------------------------------
 10    vSRX_SRIOV                     running
-      vSRX_00                        shut off
 -     vSRX_01                        shut off
 -     vSRX_Int_Router                shut off
 -     vSRX_Node0                     shut off
 -     vSRX_Node1                     shut off
```

Step 2: List all the networks configured on your host:

```
root@LabHost:~# virsh net-list --all
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 ctrl_net             active     yes           yes
 default              active     yes           yes
 fab_net              active     yes           yes
 green_net            active     yes           yes
 red_net              active     yes           yes
```

Step 3: List all the networks bound to a VM:

```
root@LabHost:~# virsh domiflist vSRX_00
Interface  Type       Source     Model       MAC
-------------------------------------------------------
-          network    default    virtio      52:54:00:42:61:aa
-          network    green_net  virtio      52:54:00:0f:77:62
-          network    red_net    virtio      52:54:00:de:39:25
```

Have you noticed that the Interface column is empty in the above output? Do you know why? Answer: Check the VM state in the output sample showed in Step 1.

Step 4: Check the XML output of a VM:

```
root@LabHost:~# virsh dumpxml vSRX_00
<domain type='kvm'>
  <name>vSRX_00</name>
  <uuid>9d69d7b2-42d8-2092-94e4-2d7b1df283ea</uuid>
  <memory unit='KiB'>4194304</memory>
```

```
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>2</vcpu>
  <os>
….
<…. skipped>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

Step 5: Explore the virsh world yourself:

```
root@LabHost:~# virsh --help
virsh [options]... [<command_string>]
virsh [options]... <command> [args...]
  options:
    -c | --connect=URI      hypervisor connection URI
    -r | --readonly         connect readonly
    -d | --debug=NUM        debug level [0-4]
    -h | --help             this help
<….Truncated>
```

More on BRCTL

BRCTL stands for bridge control. In Linux, this utility is used for Ethernet bridge administration tasks such as setting up, maintaining, and inspecting the bridge configuration. To do it yourself, hop onto your host to explore the `brctl` command:

Step 1: Check that the bridge is configured on the host:

```
root@LabHost:~# brctl show
bridge name     bridge id               STP enabled     interfaces
ctrl_net        8000.52540002c30c       yes             ctrl_net-nic
fab_net         8000.525400e8f115       yes             fab_net-nic
green_net       8000.525400260e78       yes             green_net-nic
red_net         8000.52540088230e       yes             red_net-nic
virbr0          8000.246e96a97150       no              eno1
                                                        vnet0
```

NOTE At the time this output was collected, the host had only one VM running, vSRX_SRIOV, which has only one virtio interface (default virtual network) and the two others that are bypass interfaces to the pNIC. Hence, you see only one virtual interface 'vnet0' attached to virbr0.

Try it yourself: Spin up other VMs and observe the difference it makes to the above output.

Step 2: Add a physical NIC to a bridge:

```
root@LabHost:~# brctl addif red_net eno4
root@LabHost:~# brctl show
bridge name     bridge id               STP enabled     interfaces
ctrl_net        8000.52540002c30c       yes             ctrl_net-nic
fab_net         8000.525400e8f115       yes             fab_net-nic
```

```
green_net       8000.525400260e78       yes             green_net-nic
red_net         8000.246e96a97155       yes             eno4
                                                        red_net-nic
virbr0          8000.246e96a97150       no              eno1
                                                        vnet0
```

NOTE As shown in the above example, the network red_net now has its connection to the physical NIC. With correct configuration on the device connected via interface eno4, VMs connected via virtual network (red_net) can communicate with the outside network.

Step 3: Explore the `brctl` world yourself:

```
root@LabHost:~# brctl --help
Usage: brctl [commands]
commands:
        addbr           <bridge>                add bridge
        delbr           <bridge>                delete bridge
        addif           <bridge> <device>       add interface to bridge
        delif           <bridge> <device>       delete interface from bridge
        hairpin         <bridge> <port> {on|off}        turn hairpin on/off
        setageing       <bridge> <time>         set ageing time
<….Truncated>
```

Routing Engine and Packet Forwarding Engine on vSRX

What is a Routing Engine and a Packet Forwarding Engine? Why do they need to communicate? How do we check the state of these two?

These two terms are also known as the control plane and the data plane, respectively. A Routing Engine performs any control tasks like routing adjacency and ARP learning, and feeds the Packet Forwarding Engine with information for it to continue working at peak speeds.

Since this is a book about security, let's talk about session. A stateful device like vSRX needs to keep track of the traffic that passes through it to confirm the validity of that traffic.

Taking an example of a TCP connection, once the first SYN packet arrives at vSRX, it is sent to the flow-daemon in the Packet Forwarding Engine to perform the basic checks (policy checks, routing check, firewall check, etc.). Once allowed, the packet has to be sent to the destination. Before that, the destination IP address ARP has to be resolved by the Routing Engine. If the Routing Engine is unable to send the ARP Request, or the destination does not revert with an ARP reply, the packet is dropped. If the Routing Engine receives a reply, it updates the Packet Forwarding Engine with the information, the packet is sent safely to the destination, and a session is installed.

Let's check the Routing Engine and Packet Forwarding Engine on a vSRX:

Step 1: Connect to the Routing Engine.

Upon login to the vSRX VM, you first connect to the BSD shell, and when you type `cli`, you are connected to the Routing Engine of the VM. When you type `show chassis routing-engine` on the Routing Engine, you see the details about the Routing Engine as shown here:

```
root@LabHost:~# virsh console vSRX_SRIOV
Connected to domain vSRX_SRIOV
Escape character is ^]
login: root
Password:
Last login: Sun Jan 20 00:49:39 from 172.29.186.31

--- JUNOS 18.4R1.8 Kernel 64-bit XEN JNPR-11.0-20181207.6c2f68b_2_bu
root@:~ #
root@:~ # cli
root> show chassis routing-engine
Routing Engine status:
    Total memory              4050 MB Max  3119 MB used ( 77 percent)
      Control plane memory    4050 MB Max  3119 MB used ( 77 percent)
    5 sec CPU utilization:
      User                       0 percent
      Background                 0 percent
      Kernel                     0 percent
      Interrupt                  0 percent
      Idle                      99 percent
    Model                        VSRX RE
    Start time                   2019-01-19 06:26:33 UTC
    Uptime                       23 hours, 52 minutes, 14 seconds
    Last reboot reason           Router rebooted after a normal shutdown.
    Load averages:               1 minute   5 minute  15 minute
                                     1.19       1.11       1.08
```

NOTE If you are able to connect to the operational mode and execute the commands, this means that the Routing Engine is running.

Step 2: Check the status of the Packet Forwarding Engine:

```
root> show chassis fpc pic-status
Slot 0   Online       FPC
  PIC 0  Online       VSRX DPDK GE
```

NOTE If the FPC continues to be offline, you need to check the messages or chassisd logs to get more information about the cause. How to see the log files is explained later in this chapter.

Step 3: To connect to shell, type "start shell" on configuration mode:

```
root@vSRX_00> start shell
root@vSRX_00:~ # uname -a
FreeBSD vSRX_00 JNPR-11.0-20181207.6c2f68b_2_bu FreeBSD JNPR-11.0-20181207.6c2f68b_2_builder_
stable_11 #0 r356482+6c2f68b(HEAD): Thu Dec  6 21:49:22 PST 2018     builder@feyrith.juniper.net:/
volume/build/junos/occam/llvm-5.0/freebsd/stable_11/20181113.154712_2_builder_stable_11.6c2f68b/obj/
amd64/juniper/kernels/JNPR-AMD64-
```

NOTE This is vSRX UNIX shell and you can navigate through the directories.

Step 4: To check which processes are running on the vSRX VM, connect to the shell and type "ps –aux":

```
root@vSRX_00:~ # ps -aux
USER    PID  %CPU %MEM     VSZ     RSS TT  STAT STARTED      TIME COMMAND
root     11 144.3  0.0       0      32  -  RL   23:32   822:57.23 [idle]
root   5364  59.3 65.3 2807508 2709892  -  S    23:34   437:02.42 /usr/sbin/srx
```

Step 5: Check the processes running from the CLI:

```
root@vSRX_00> show system processes extensive
last pid:  6198;  load averages:  1.34,  1.33,  1.29  up 0+10:43:09    10:16:01
280 processes: 9 running, 249 sleeping, 22 waiting

Mem: 38M Active, 693M Inact, 2918M Wired, 56M Buf, 270M Free
Swap: 1024M Total, 1024M Free


  PID USERNAME PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
   11 root     155 ki31     0K    32K RUN     0 603:30  94.48% idle{idle: cpu0}
   11 root     155 ki31     0K    32K CPU1    1 220:51  51.56% idle{idle: cpu1}
 5364 root      88    0  2742M  2646M CPU1    1 421:51  51.17% srxpfe{lcore-slave-1}
 5364 root      21    0  2742M  2646M RUN     0  15:44   2.10% srxpfe{srxpfe}
 5443 root      20    0 94748K 17220K RUN     0   5:41   0.29% llmd{llmd}
 5660 root      20    0   822M 44500K RUN     0   3:16   0.10% authd
 5534 root      20    0   737M 32864K select  0   0:00   0.10% mgd
```

Understanding Packet Walk and Taps

Have you ever been to a running event? How do the organizers ensure that the athletes run the whole distance and in a certain amount of time?

The race route is generally predefined, and a transponder working on a radio-frequency identification (RFID) basis is attached to the athlete. It emits a unique code that is detected by radio receivers located at strategic points throughout the event.

Consider traffic/packets as a runner and virtual networks (vNIC) as the strategic points where tapping can confirm if the runner (traffic/packet) is reached the particular vNIC or not. Let's use the following topology to try and understand the flow:

```
vSRX_00 ---vnet_x --- red_net --- vnet_x ----vSRX_01
```

When you initiate a ping from vSRX_00 to vSXR_0, what all needs to be in order for the ping to work?

- vSRX_00 – Correct interface IP address and zone configuration
- vSRX_00 – Interface connected to the red_net
- red_net – Network showing the connected MAC address of the above interface

- vSRX_01 – Correct interface IP address and zone configuration
- vSRX_00 – Interface connected to the red_net
- red_net – Network showing the connected MAC address of the above interface

To do it yourself: Jump in and do some packet dumps. Use the following steps:

Step 1: Start the vSRX_00 and vSRX_01 VM if they are shut off:

```
virsh start vSRX_00
virsh start vSRX_01
“virsh list –all” should show the VM as ‘running’
```

Step 2: Once the VMs boot up, log in and check that the interfaces are up using the following command from the operational mode:

```
root@vSRX_00> show interface terse
```

Step 3: Confirm that the interface configuration is accurate:

```
vSRX_00 – ge-0/0/1 ip should be 172.16.10.1
vSRX_01 – ge-0/0/1 ip should be 172.16.10.2
```

Step 4: Check the MAC address on the vSRX VM:

```
root@vSRX_00> show interfaces ge-0/0/1 | match Current
  Current address: 52:54:00:de:39:25, Hardware address: 52:54:00:de:39:25

root@vSRX_01> show interfaces ge-0/0/1 | match Current
  Current address: 52:54:00:c0:c6:d6, Hardware address: 52:54:00:c0:c6:d6
```

Step 5: Check that the virtual network has the correct MAC address on the connected virtual NIC:

```
root@LabHost:~# brctl showmacs red_net
port no mac addr                is local?       ageing timer
  2     24:6e:96:a9:71:55       yes                0.00
  1     52:54:00:88:23:0e       yes                0.00
  4     52:54:00:c0:c6:d6       no                59.69
  3     52:54:00:de:39:25       no                59.69
  4     fe:54:00:c0:c6:d6       yes                0.00
  3     fe:54:00:de:39:25       yes                0.00
```

Step 6: Confirm that the correct port number connected to the VM by using the virsh command as following:

```
root@LabHost:~# virsh domiflist vSRX_00
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet2      network    default    virtio      52:54:00:42:61:aa
vnet3      network    green_net  virtio      52:54:00:0f:77:62
vnet4      network    red_net    virtio      52:54:00:de:39:25
```

```
root@LabHost:~# virsh domiflist vSRX_01
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet5      network    default    virtio      52:54:00:00:48:2e
vnet6      network    green_net  virtio      52:54:00:3f:4f:ea
vnet7      network    red_net    virtio      52:54:00:c0:c6:d6
```

As shown in the sample above, `vnet4` and `vnet7` are the virtual interfaces where you can tap into the traffic.

Step 7: Capture the traffic. Open two separate SSH sessions to the host and start capturing using the tcpdump utility in Linux:

```
root@LabHost:~# tcpdump -nni vnet4 icmp
root@LabHost:~# tcpdump -nni vnet7 icmp
```

Step 8: Open another SSH to the host and console log in to vSRX_00 and send two ICMP ping to 172.6.10.2:

```
root@vSRX_00> ping 172.16.10.2 count 2
PING 172.16.10.2 (172.16.10.2): 56 data bytes
64 bytes from 172.16.10.2: icmp_seq=0 ttl=64 time=1.097 ms
64 bytes from 172.16.10.2: icmp_seq=1 ttl=64 time=1.184 ms

--- 172.16.10.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 1.097/1.140/1.184/0.044 ms
```

Step 9: Use the tcpdump utility to show the ICMP echo and reply:

```
root@LabHost:~# tcpdump -nni vnet4 icmp
tcpdump: WARNING: vnet4: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on vnet4, link-type EN10MB (Ethernet), capture size 65535 bytes
13:25:25.684932 IP 172.16.10.1 > 172.16.10.2: ICMP echo request, id 48405, seq 0, length 64
13:25:25.685348 IP 172.16.10.2 > 172.16.10.1: ICMP echo reply, id 48405, seq 0, length 64
13:25:26.685234 IP 172.16.10.1 > 172.16.10.2: ICMP echo request, id 48405, seq 1, length 64
13:25:26.685867 IP 172.16.10.2 > 172.16.10.1: ICMP echo reply, id 48405, seq 1, length 64
root@LabHost:~# tcpdump -nni vnet7 icmp
tcpdump: WARNING: vnet7: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on vnet7, link-type EN10MB (Ethernet), capture size 65535 bytes
13:25:25.684955 IP 172.16.10.1 > 172.16.10.2: ICMP echo request, id 48405, seq 0, length 64
13:25:25.685328 IP 172.16.10.2 > 172.16.10.1: ICMP echo reply, id 48405, seq 0, length 64
13:25:26.685249 IP 172.16.10.1 > 172.16.10.2: ICMP echo request, id 48405, seq 1, length 64
13:25:26.685839 IP 172.16.10.2 > 172.16.10.1: ICMP echo reply, id 48405, seq 1, length 64
```

NOTE If you do not see ICMP echo on vnet4 capture, you need to "`monitor traffic interface ge-0/0/1`" on the vSRX VM to check if ICMP is being crafted.

Likewise, in case you have a physical NIC connected to a virtual network and the traffic is destined to exit out, you need to initiate the tcpdump utility on the physical interface from the host.

Understanding Host and vSRX VM Logs

One of the reasons we like Linux and consider it a great operating system is that anything and everything happening on and to the system is being logged in some manner. This information is very important and is invaluable when one is troubleshooting a problem.

On Ubuntu, or on vSRX, logging is saved in the traditional system subdirectory /var/log.

Both Linux (Ubuntu) and vSRX are open to allow the user to configure in order to write a certain type of log to a specific file.

Here is a list of a few default log files that could be helpful while troubleshooting a problem:

Host Log Files

VM Info

On an Ubuntu host, under "/var/log/libvirt/qemu" path, you'll see files for each VM that you spin.

The file name is the VM name followed by an extension '.log'.

Do it yourself: navigate to the directory and see the log files for your VMs.

If the VM does not start up upon using the virsh command, the error generated for the non-starting VM is listed here in the following log files:

kern.log

As the name suggests any information relating to the kernel running on your host can be seen from here.

syslog

What you have as a messages file in vSRX is a syslog file in Linux (Ubuntu). Consult the system log when you can't locate the desired log information in another log.

vSRX VM Logs

messages

You can view the system messages in the log files with the '`show log messages`' command. Information about any daemon or any configuration change or any event is logged in this file. To view the log messages, type the following command from a vSRX VM operational mode:

```
root>show log messages
root> show log messages
Jan 19 12:30:38   eventd[20226]: SYSTEM_ABNORMAL_SHUTDOWN: System abnormally shut down
Jan 19 12:30:38   eventd[20226]: SYSTEM_OPERATIONAL: System is operational
Jan 19 12:30:38   kernel: Copyright (c) 1992-2017 The FreeBSD Project.
Jan 19 12:30:38   kernel: Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
Jan 19 12:30:38   kernel:       The Regents of the University of California. All rights reserved.
Jan 19 12:30:38   kernel: FreeBSD is a registered trademark of The FreeBSD Foundation.
```

Chassisd Logs

Chassisd log files display details of the chassis-control process such as events and information relating to hardware, chassis control, and related logs. You can also find information related to the Packet Forwarding Engine in chassisd logs.

You can see the logs related to Routing Engine-Packet Forwarding Engine connection here:

```
root> show log chassisd
Jan 19 12:30:40  FPC in slot 0 goes online
Jan 19 12:30:40  fru_power_on_state_timer: FPC 0 step 0
Jan 19 12:30:40  FPC 0 power on in 7 sec
Jan 19 12:30:40  alarmd connection completed
Jan 19 12:30:40  send: clear all chassis class alarms
Jan 19 12:30:47  fru_power_on_state_timer: FPC 0 step 1
Jan 19 12:30:47  Power on FPC 0
Jan 19 12:30:47 CHASSISD_IPC_WRITE_ERR_NULL_ARGS: FRU has no connection arguments fru_send_msg FWDD
Jan 19 12:30:47  send: fwdd, fpc 0 powered on
Jan 19 12:30:47  setup_power_on_timeout FPC timeout 300 secs
Jan 19 12:31:05  ch_ipc_connect:Setting TCP keepalive count to 5.

Jan 19 12:31:05  CMLC: Chasd TCP socket KeepIdle=1000,KeepInterval=1000 KeepCount=5
```

Jsrpd (Juniper Services Redundancy Protocol Daemon)

Information about the chassis cluster, such as events leading to the redundancy feature of Junos, is available in this log file:

```
root> show log jsrpd
Jan 19 12:30:39 JSRPD release 18.4R1.8 built by builder on 2018-12-
17 04:35:07 UTC starting, pid 20289
Jan 19 12:30:39 node id invalid, cluster-id 0 in kernel
```

Since the chassis cluster is not enabled on the node from where this output is taken, the node ID we see is invalid. Connect to the VM vSRX_Node0 VM to check the chassis cluster log. Execute an RG-1 failover using the following command and check the log messages in jsrpd logs:

```
root> request chassis cluster failover redundancy group 1 node 1
```

Interactive-commands

This log file gives information about all the commands that a user runs on the CLI. This command is handy in an event when you need to understand all commands that were run by different users, or by you in the past:

```
root> show log interactive-commands | last 5
Jan 20 10:31:25   mgd[22399]: UI_CMDLINE_READ_
LINE: User 'root', command 'show log chassisd | last 100 '
Jan 20 10:33:07   mgd[22399]: UI_CMDLINE_READ_LINE: User 'root', command 'show log chassisd '
Jan 20 10:35:34   mgd[22399]: UI_CMDLINE_READ_LINE: User 'root', command 'show log jsrpd '
Jan 20 10:38:44   mgd[22399]: UI_CMDLINE_READ_LINE: User 'root', command 'show log interactive-
commands '
Jan 20 10:38:52   mgd[22399]: UI_CMDLINE_READ_LINE: User 'root', command 'show log interactive-
commands | last 5 '
```

Issues You Might Face While Installing vSRX

- VM boots but stuck in boot process and shows kernel panic error

Check to see if the boot file is correct. Upon download of the image file always check the checksum and compare the same with the one provided on the website.

- FPC-PIC status stuck in present or Offline state

Use the following CLI command to check the status:

```
root> show chassis fpc pic-status
Slot 0   Present      FPC
```

Check "chassisd" logs in vSRX to confirm cause of the FPC stuck in Present state.

- Error: "Host does not support passthrough of pci device."

As stated in the error message, mostly this error is relative to SR-IOV not enabled either in BIOS or on GRUB. Follow the steps in the lab exercise carefully:

```
Error: "Error starting domain: Device 0000:01:10.1 not found: could not access /sys/bus/pci/
devices/0000:01:10.1/config: No such file or directory"
```

This error indicates that the number of virtual functions are not defined in the file sriov_numvfs.

- vSRX with SR-IOV interfaces does not show up if the host interface is down.

Reference: https://kb.juniper.net/KB31894. This is as per design as the DPDK link level status detecting API is not implemented for the SR-IOV physical function.

Summary

Now that you have learned how to build and configure vSRX VMs, why not go ahead and deploy the vSRX to meet your own specific requirements? The virtual SRX enables you to quickly introduce a new service or sandbox-test a new configuration.

The vSRX supports Layer 2 to Layer 7 technologies that are available on a physical SRX. However, it is always best to check the supported and unsupported features on a vSRX version by checking the release notes here: https://www.juniper.net/documentation/product/en_US/vsrx.

If you're looking for new use cases for vSRX, try these:

- Private cloud – vSRX maximizes resources by pooling and sharing along with managing functional separation to keep the data private.
- Public and hybrid cloud – vSRX VM dedicated to each customer as an IPsec end point.
- vCPE solution – vSRX dedicated for a specific feature such as UTM for one customer and IPS+AppSecure for another customer.
- uCPE solution – vSRX running on a Universal CPE chassis such as NFX platforms.
- Virtual environment – vSRX acting as a security path for VMs on a server.
- Sandbox – vSRX allows you to create a Junos sandbox with vSRX in your lab environment to allow you to create network simulation and configuration testing.

Appendix

This Appendix contains assorted bits of useful information that didn't fit in the step-by-step instructions within the book, including the 10 Most Active vSRX Support Issues (and the links to their answers).

Installing vSRX with virt-manager (GUI Package in Linux)

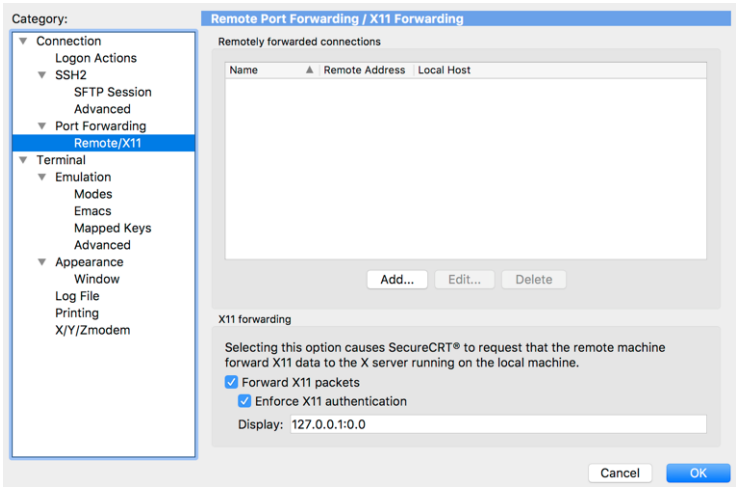
In Linux, you utilized virsh to create, console, manage, and destroy VMs. Linux offers another utility which has a graphical interface for the user.

Checklist for using the virt-manager utility:

- X11 Forwarding enabled on your text editor
- Virt-manager program installed on the host

X11 Forwarding requires you to check your connection settings in a text editor and enable the feature while adding your display as 127.0.0.1:0.0 (that is your local machine). Here's an example for enabling X11 forwarding using Secure CRT terminal:

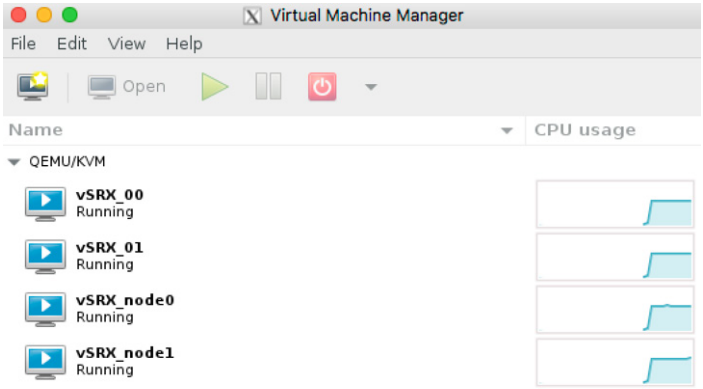
Open an SSH session in SecureCRT and navigate to Connection > Port-Forwarding > Remote/X11.



Recall you have already installed the virt-manager package in Chapter 2. Now log in to the host and complete the following steps:

```
root@LabHost:~# virt-manager
```

A new GUI window pops up as shown .

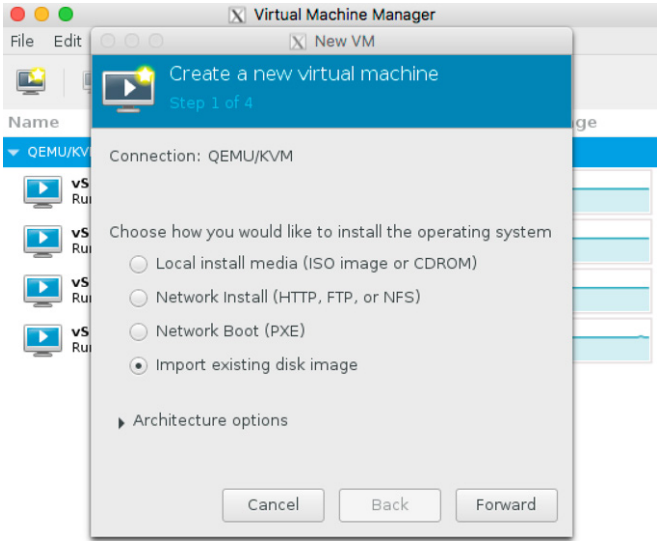


The Virtual Machine Manager window lists all the VMs that are configured on the host and displays their status. To instantiate a VM using virt-manager follow these steps.

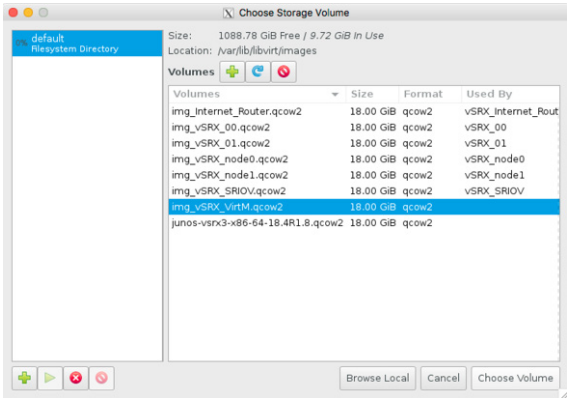
Step 1: Copy the vSRX image and save it with a new name to create a VM using virt-manager:

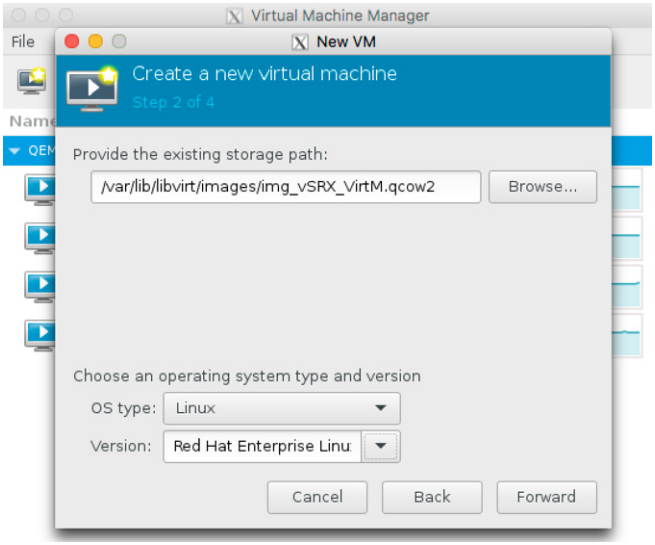
```
root@LabHost:~# cp /var/lib/libvirt/images/junos-vsrx3-x86-64-18.4R1.8.qcow2 /var/lib/libvirt/
images/img_vSRX_VirtM.qcow2
root@LabHost:~# ls -la /var/lib/libvirt/images/
total 4822560
drwx--x--x 2 root         root      4096 Jan 28 04:05 .
drwxr-xr-x 7 root         root      4096 Jan 26 06:36 ..
-rw-r--r-- 1 libvirt-qemu kvm  866189312 Jan 28 04:05 img_vSRX_00.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  862453760 Jan 28 04:05 img_vSRX_01.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  875560960 Jan 28 04:05 img_vSRX_node0.qcow2
-rw-r--r-- 1 libvirt-qemu kvm  869466112 Jan 28 04:05 img_vSRX_node1.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 28 04:05 img_vSRX_VirtM.qcow2
-rw-r--r-- 1 root         root 732299264 Jan 26 05:22 junos-vsrx3-x86-64-18.4R1.8.qcow2
```

Step 2: Click on the Create New Virtual Machine icon and select the Import existing disk image. Click Forward.

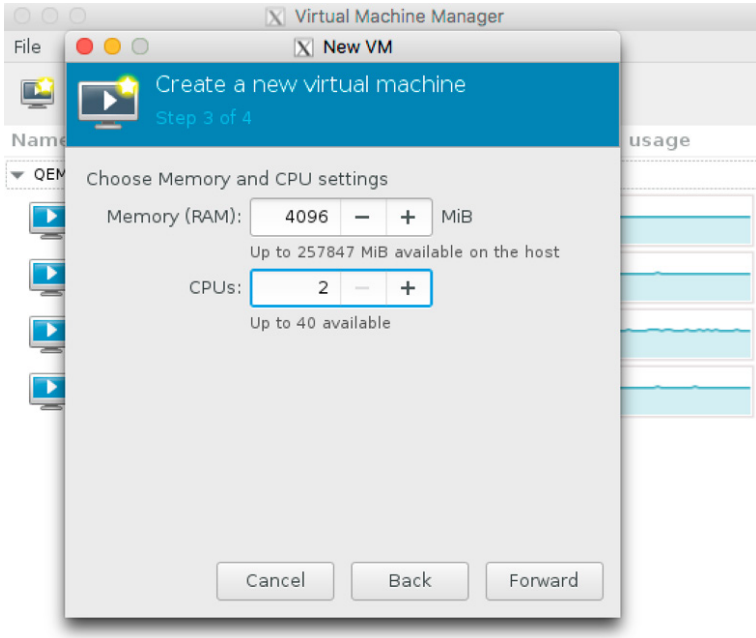


Step 3: Browse to the location of the downloaded vSRX image and select the vSRX image. Click on the select volume. Next, select Linux from the OS type list and select Red Hat Enterprise Linux 7 from the expanded Version list and click Forward.

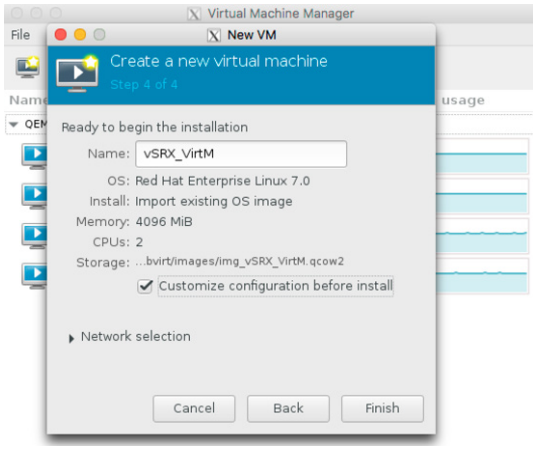




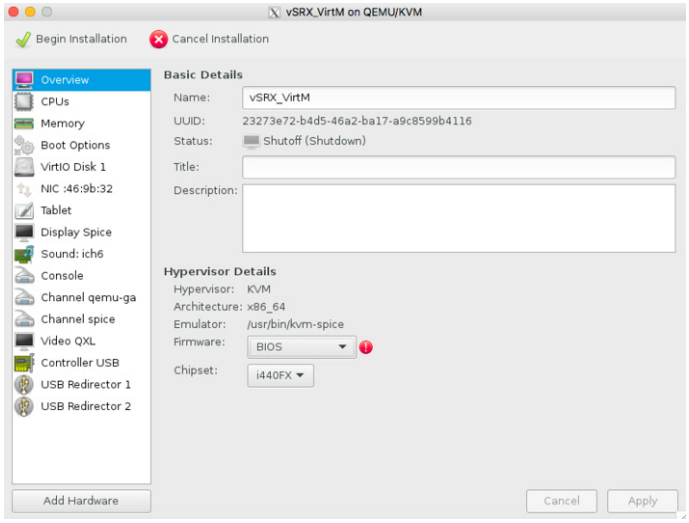
Step 5: Select RAM as 4096 and CPU as 2. Click Forward.



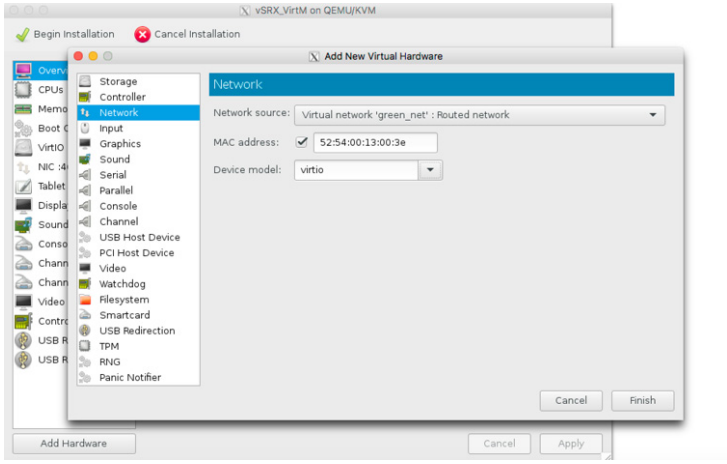
Step 6: Name the VM as vSRX_VirtM and checkmark Customize configuration before install. Click Finish.



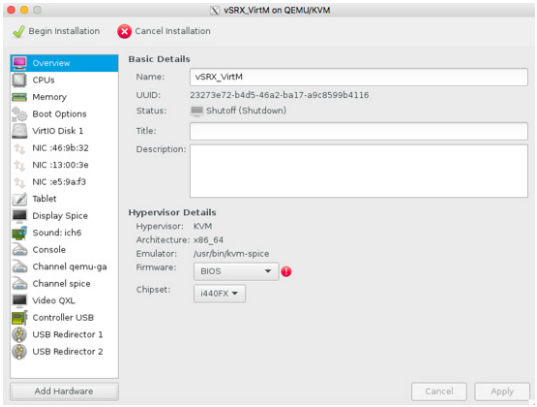
Step 7: Use the following screen to review more details about the VM.



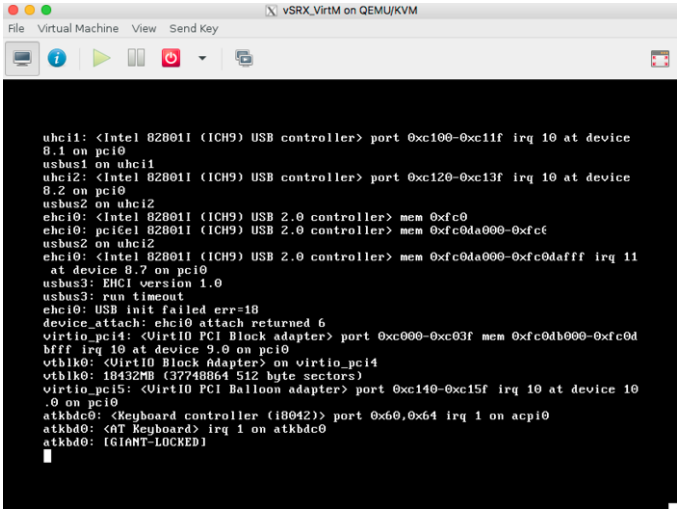
Step 8: Click Add Hardware and select Network. Next select green_net network in Network Source and select virtio in device model. Click Finish.



Step 9: Follow Step 8 to add the red_net network also. Now, on the left side, you'll see the three NICs present.



Step 10: Click Begin Installation. A console window opens for the new VM and displays the status of the installation.



The VM manager creates and launches the vSRX VM. This is how you configure a VM using virt-manager.

Scaling Up a vSRX Instance

Scaling up a vSRX VM means increasing the throughput for processing traffic. The base seeds are the memory and the CPU. For all the VMs that have spawned, we pool 4GB as the memory and two vCPUs as the CPU. If one looks at the `show chassis hardware` output clearly, the vSRX VM runs as an S (*small*) variant:

```
root@vSRX_SRIOV> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                4c7bd984b97f      VSRX
Midplane
System IO
Routing Engine                                            VSRX-S   <<<<<<
FPC 0                                                     FPC
  PIC 0                                                   VSRX DPDK GE
Power Supply 0
root@vSRX_SRIOV> show chassis routing-engine
Routing Engine status:
    Total memory              4050 MB Max  3159 MB used ( 78 percent)
      Control plane memory    4050 MB Max  3159 MB used ( 78 percent)

root@vSRX_SRIOV> show system processes extensive
last pid: 23045;  load averages:  1.22,  1.21,  1.20  up 9+00:23:14    10:58:01
279 processes: 3 running, 254 sleeping, 22 waiting

Mem: 30M Active, 642M Inact, 2926M Wired, 60M Buf, 320M Free
Swap: 1024M Total, 28K Used, 1024M Free


 PID USERNAME PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
11 root     155 ki31     0K    32K RUN     0 204.5H 100.00% idle{idle: cpu0}
11 root     155 ki31     0K    32K RUN     1  79.5H  55.47% idle{idle: cpu1}
12368 root      52    0  2740M  2647M nanslp  1 136.8H  50.68% srxpfe{lcore-slave-1}
12368 root      21    0  2740M  2647M select  0 340:27   2.10% srxpfe{srxpfe}
12490 root      20    0 94748K 13752K nanslp  0 113:26   0.39% llmd{llmd}
```

Scaling up means scaling the performance and capacity of a vSRX instance by increasing the number of vCPUs, or the amount of vRAM allocated to the vSRX. Simple!

So, to scale up a vSRX VM to an M variant, we define the vCPU to 5 and memory to 8GB. When you define five vCPUs, one vCPU is required for the Routing Engine and the remaining four are propagated for the Packet Forwarding Engine. You need to configure network multi-queuing to support an increased number of vCPUs for the data plane (Packet Forwarding Engine).

This setting updates the libvirt driver to enable multi-queue virtio-net to scale up the network performance as the number of vCPUs increases. We need to edit the XML file of the VM to scale up the vSRX_01 VM into an M (*medium*) variant.

The changes required in the XML file are:

- vCPUs from 2 to 5
- Memory from 4194304 to 8388608
- Add this in the interface configuration to update the queue, at the <driver name='vhost' queues='x'/> line change as "<driver name='vhost' queues='8'/>"

To change the XML file, follow these steps.

Step 1: Edit the file using the following command:

```
root@LabHost:~# virsh edit vSRX_01
Change the following:
  <name>vSRX_01</name>
  <uuid>6c02b04a-2e47-4add-b1ed-38fbe35bc192</uuid>
  <memory unit='KiB'>4194304</memory>
  <currentMemory unit='KiB'>4194304</currentMemory>
  <vcpu placement='static'>2</vcpu>
with;
  <name>vSRX_01</name>
  <uuid>6c02b04a-2e47-4add-b1ed-38fbe35bc192</uuid>
  <memory unit='KiB'>8388608</memory>
  <currentMemory unit='KiB'>8388608</currentMemory>
  <vcpu placement='static'>5</vcpu>
```

Step 2: Add the following line for the green_net and red_net interfaces:

```
<driver name='vhost' queues='8'/>
```

Step 3: Add the previous line after "model type" like this:

```
    <interface type='network'>
      <mac address='52:54:00:67:34:eb'/>
      <source network='blue_net'/>
      <model type='virtio'/>
      <driver name='vhost' queues='8'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </interface>
    <interface type='network'>
      <mac address='52:54:00:90:c9:18'/>
      <source network='red_net'/>
      <model type='virtio'/>
      <driver name='vhost' queues='8'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </interface>
```

Now you can save the file.

Step 4: Start the VM after saving the file:

```
root@LabHost:~# virsh start vSRX_01
root@LabHost:~# virsh list --all
```

```
 Id    Name                           State
----------------------------------------------------
 1     vSRX_SRIOV                     running
 2     vSRX_01                        running
 -     vSRX_00                        shut off
 -     vSRX_Internet_Router           shut off
 -     vSRX_node0                     shut off
 -     vSRX_node1                     shut off
```

Step 5: Log in to the VM and check the hardware, CPU, and memory details:

```
root@vSRX_01> show chassis hardware
Hardware inventory:
Item             Version  Part number  Serial number     Description
Chassis                                38fbe35bc192      VSRX
Midplane
System IO
Routing Engine                                            VSRX-M  <<<<<
FPC 0                                                     FPC
  PIC 0                                                   VSRX DPDK GE
Power Supply 0
```

And here you can notice that the Routing Engine value has changed from `VSRX-S` to `VSRX-M`. This confirms that your VM instance is now scaled up to *medium* flavor:

```
root@vSRX_01> show chassis routing-engine
Routing Engine status:
    Total memory              8146 MB Max  7006 MB used ( 86 percent)
      Control plane memory    8146 MB Max  7006 MB used ( 86 percent)

root@vSRX_01> show system processes extensive
last pid: 15897;  load averages:  3.65,  3.58,  3.54  up 8+23:45:44    10:59:56
303 processes: 13 running, 265 sleeping, 25 waiting

Mem: 32M Active, 679M Inact, 6727M Wired, 129M Buf, 478M Free
Swap: 1024M Total, 1024M Free


  PID USERNAME PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
   11 root     155 ki31     0K    80K RUN     0 204.3H  94.68% idle{idle: cpu0}
 5366 root      52    0  6324M  6231M RUN     4 140.5H  64.99% srxpfe{lcore-slave-4}
 5366 root      92    0  6324M  6231M CPU1    1 138.5H  64.79% srxpfe{lcore-slave-1}
   11 root     155 ki31     0K    80K CPU3    3 111.4H  53.27% idle{idle: cpu3}
   11 root     155 ki31     0K    80K CPU2    2 111.0H  52.78% idle{idle: cpu2}
 5366 root      52    0  6324M  6231M RUN     2 104.8H  49.37% srxpfe{lcore-slave-2}
 5366 root      52    0  6324M  6231M nanslp  3 104.4H  48.88% srxpfe{lcore-slave-3}
   11 root     155 ki31     0K    80K RUN     1  77.3H  38.48% idle{idle: cpu1}
   11 root     155 ki31     0K    80K RUN     4  75.3H  37.50% idle{idle: cpu4}
 5366 root      21    0  6324M  6231M RUN     0 308:22   1.66% srxpfe{srxpfe}
 5429 root      20    0   101M 17384K RUN     0 117:39   0.39% llmd{llmd}
```

Attach a New Network or Add an Interface to an Existing vSRX VM

Let's say a need arises where you want to add another interface to a running vSRX VM. You wouldn't recreate the VM from scratch but just add another interface. So let's add a new network to vSRX_00 by using the following steps:

Step 1: First we need to create a new network, create a XML file, then define and start the network:

```
root@LabHost:~# nano /etc/libvirt/qemu/networks/new_net.xml
<network>
  <name>blue_net</name>
  <forward mode='route'/>
  <bridge name='blue_net' stp='on' delay='0'/>
  <ip address='192.168.125.1' netmask='255.255.255.0'>
    <dhcp>
     <range start='192.168.125.100' end='192.168.125.250'/>
    </dhcp>
  </ip>
</network>

root@LabHost:~# virsh net-define /etc/libvirt/qemu/networks/new_net.xml
Network new_net defined from /etc/libvirt/qemu/networks/new_net.xml
root@LabHost:~# virsh net-start new_net
Network new_net started
root@LabHost:~# virsh net-autostart new_net
Network new_net marked as autostarted
```

Step 2: To check that the network is installed:

```
root@LabHost:~# virsh net-list --all
Name                 State      Autostart     Persistent
----------------------------------------------------------
 blue_net             active     yes           yes
 ctrl_net             active     yes           yes
 default              active     yes           yes
 fab_net              active     yes           yes
 green_net            active     yes           yes
 new_net              active     yes           yes
 red_net              active     yes           yes
```

Step 3: To check what networks are already connected to vSRX_00:

```
root@LabHost:~# virsh domiflist vSRX_00
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet4      network    default    virtio      52:54:00:86:82:c2
vnet5      network    green_net  virtio      52:54:00:25:e1:06
vnet6      network    red_net    virtio      52:54:00:9f:5e:6e
```

Step 4: To connect the new network to a running VM:

```
root@LabHost:~# virsh attach-interface --domain vSRX_00 --type bridge --source new_
net --model virtio
Interface attached successfully
```

Step 5: For vSRX to install said network, a reboot is required:

```
root@LabHost:~# virsh console vSRX_00
Connected to domain vSRX_00
Escape character is ^]
root@vSRX_00> request system reboot
Reboot the system ? [yes,no] (no) yes
```

Step 6: To check that the interface is installed, check using the `virsh` command.

Step 7: And, to check that the vSRX has the interface, check using CLI command:

```
root@vSRX_00> show interfaces terse
Interface               Admin Link Proto    Local                 Remote
ge-0/0/0                up    up
ge-0/0/0.0              up    up   inet     192.168.10.1/24
gr-0/0/0                up    up
ip-0/0/0                up    up
lsq-0/0/0               up    up
lt-0/0/0                up    up
mt-0/0/0                up    up
sp-0/0/0                up    up
sp-0/0/0.0              up    up   inet
                                   inet6
sp-0/0/0.16383          up    up   inet
ge-0/0/1                up    up
ge-0/0/1.0              up    up   inet     172.16.10.1/30
ge-0/0/2                up    up
```

Challenge Lab I: Convert Topology 1 and create an IPsec tunnel between vSRX_00 and vSRX_01.

Challenge Lab II: Convert Topology 3 and create an IPsec tunnel between Branch and HQ.

Challenge Lab III: Add a dynamic routing protocol over IPsec connections created in Challenge Labs I & II.

Most Popular vSRX Support Issues

Loading an Initial Configuration on vSRX:
https://www.juniper.net/documentation/en_US/vsrx/topics/task/configuration/security-vsrx-kvm-bootstrap-config.html.

Using Cloud-Init for the configuration:
https://www.juniper.net/documentation/en_US/vsrx/topics/task/configuration/security-vsrx-cloud-init-support.html.

Difference between vSRX2.0 and vSRX3.0:
https://kb.juniper.net/InfoCenter/index?page=content&id=KB33572.

SkyATP with vSRX:
https://kb.juniper.net/InfoCenter/index?page=content&id=KB31787.

New MAC address derivation after chassis cluster is enabled:
https://kb.juniper.net/InfoCenter/index?page=content&id=KB33244.

Interface Naming and mapping in vSRX Chassis Cluster mode:
https://www.juniper.net/documentation/en_US/vsrx/topics/reference/general/security-vsrx-interface-names.html.

Memory utilization calculation and description of different types of memory:
https://kb.juniper.net/InfoCenter/index?page=content&id=KB32247.

vSRX interfaces did not go down when assigned to SR-IOV virtual functions:
https://kb.juniper.net/InfoCenter/index?page=content&id=KB31894.

vSRX Session Scaling:
https://www.juniper.net/documentation/en_US/vsrx/topics/concept/security-vsrx-kvm-understanding.html#jd0e147.

Can a vSRX license be transferred to a new instance?
https://kb.juniper.net/KB33211.

Recovering root password for vSRX on KVM?
https://www.juniper.net/documentation/en_US/vsrx/topics/task/multi-task/security-vsrx-kvm-root-password-recovery.html.

vSRX Feature License:
https://www.juniper.net/documentation/en_US/vsrx/topics/concept/security-vsrx-feature-licenses-overview.html.

Managing License:
https://www.juniper.net/documentation/en_US/vsrx/topics/task/multi-task/security-vsrx-license-managing.html.

vSRX License Model Numbers:
https://www.juniper.net/documentation/en_US/vsrx/topics/reference/general/security-vsrx-feature-licenses.html.