JUNIPER NETWORKS | Engineering Simplicity

DAY ONE: vMX UP AND RUNNING, 2ND EDITION

By Matt Dinham, Madhavi Katti, and Vishruth Reddy

DAY ONE: vMX UP AND RUNNING, 2ND EDITION

This *Day One* book follows a lab setup and configuration of Juniper's vMX running on Ubuntu Linux. If you are running VMWare there's a chapter at the end of the book to walk you through that build.

The vMX is a virtual Juniper Networks MX Series router that has been optimized to run as software on x86 servers. Like other physical MX routers, vMX runs the Junos OS, and the Trio chipset has been compiled for x86. This means the sophisticated Layer 2, Layer 2.5, and Layer 3 forwarding features of the Junos OS that work with the physical MX platform are also present on the vMX.

The vMX can be installed on any server hardware of your choice, so long as it is x86-based with an Intel Nehalem or newer generation CPU and running Linux KVM or VMware. This book focuses on a lab build of vMX 20.2R2 running on Linux KVM.

There's plenty of example configurations and tips, as well as an Appendix with the ten most common vMX Juniper Support issues and their resolution links.

"After reading this book I could effortlessly create a large service provider network using over a dozen vMX routers, despite the fact that I have never touched either KVM or the vMX before. This Day One *book on vMX provides a detailed and fun walkthrough of several scenarios, equipping you with a foundation to start building your own networks and labs. It's smart and to the point."*

Said van de Klundert, Network Automation Engineer, IBM Cloud

IT'S DAY ONE AND YOU HAVE A JOB TO DO:

- Work with the vMX architecture and be able to deploy the book's use cases.
- Understand the build, configuration, and deployment of vMX in your lab or production environments.
- Scale an instance of the vMX.
- License the vMX for a lab or production deployment.
- Troubleshoot vMX installation and deployment issues.

ISBN 978-1941441350
51600
9 781941 441350

Day One: vMX Up and Running, 2nd Edition

by Matt Dinham, Madhavi Katti, Vishruth Reddy

JUNIPER NETWORKS

Published by Juniper Networks Books
Authors: Matt Dinham, Madhavi Katti, Vishruth Reddy
Technical Reviewers: Melchior Aelmans, Ashish Gupta
Editor in Chief: Patrick Ames
Copy Editor: Nancy Koerbel

Printed in the USA by Vervante Books.

Version History: v1, June 2021
2 3 4 5 6 7 8 9 10
Comments, errata: dayone@juniper.net

About the Author
Matt Dinham is an independent consulting Network Architect based in the UK, and a Juniper Ambassador. Matt has over 20 years experience working within Enterprise and Service Provider environments (public & private sector), and is certified CCIE #16387 (R&S, SP). Find Matt on Twitter: @mattdinham.

Madhavi Katti is an Information Development Engineer at Juniper Networks with over 13 years of experience in writing and developing documentation for networking and telecommunications. Madhavi contributes to product documentation for security and virtualization products.

Vishruth Reddy is a ATAC engineer based in Bangalore, India. He has 10 years of experience working with different Juniper product lines, mainly Junos(MX & vMX), Subscriber Management, Northstar and cRPD. This is his first *Day One* but in his years of work as a Technical Support Engineer Staff, he's seen how important the series is for newbies.

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 for looking over our words and offering plenty of encouragement along the way. Special thanks to Ryan Israel for help with lab set up and to Indira Upadhayaya for her vision, support, and encouragement.

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 network administration with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow.

- Download a free PDF edition at https://www.juniper.net/dayone
- Purchase the paper edition at Vervante Books (www.vervante.com).

Key vMX Resources

The authors of this book highly recommend the following vMX resources, especially the Juniper TechLibrary and its up-to-date information and specifications. Also a list of the ten most popular tech support questions, and links to their solutions, can be found in the Appendix.

Juniper TechLibrary:
https://www.juniper.net/documentation/product/us/en/vmx/

Juniper vMX:
https://www.juniper.net/us/en/products-services/routing/mx-series/vmx/

Before Reading This Book You Need

Before reading this book, you should be familiar with the basic administrative functions of the Junos OS, including working with operational commands and reading, understanding, and changing Junos configurations. There are several books in the Day One library on learning Junos, at http://www.juniper.net/dayone.

This book makes a few assumptions about you, the reader:

- You have a basic understanding of IPv4 and the OSPF and BGP routing protocols.
- You are familiar with the Junos OS operation and configuration.
- You have a basic understanding of Linux System Administration (preferably Ubuntu), and knowledge of the Linux Virtualization solution KVM.
- You have a basic understanding of MPLS.
- For the lab build you have access to a laptop or desktop with at least 6 x CPU cores and 16-32GB RAM.

After Reading This Book, You'll Be Able To

- Work with the vMX architecture and be able to deploy the book's use cases.
- Understand the build, configuration, and deployment of vMX in your lab or production environments.
- Scale an instance of the vMX.
- License the vMX for a lab or production deployment.
- Troubleshoot vMX installation and deployment issues.

Chapter 1

Introduction to the vMX

This *Day One* book walks you through the lab set up and configuration of Juniper's vMX running on Ubuntu Linux. If you are running VMware, Chapter 4 guides you through that build. This first chapter introduces you to the architecture of the vMX, which is key to understanding its sizing and licensing models. Let's get started.

What is vMX?

The vMX is a virtual Juniper Networks MX Series router optimized to run as software on x86 servers. Like physical MX routers, vMX runs the Junos OS, and the Trio chipset has been compiled for x86. This means the sophisticated Layer 2, Layer 2.5, and Layer 3 forwarding features of the Junos OS that work with the physical MX platform are also present on the vMX.

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

This book focuses on a lab build of vMX 20.2R2 running on Linux KVM. We've also included Chapter 6 in this book to walk you through the installation of vMX on VMware's ESXi Hypervisor.

Architecture of vMX

As shown in Figure 1.1, the vMX actually consists of two separate VMs – a virtual control plane (VCP) running the Junos OS, and a virtual forwarding plane (VFP) running the virtualized Trio forwarding plane.

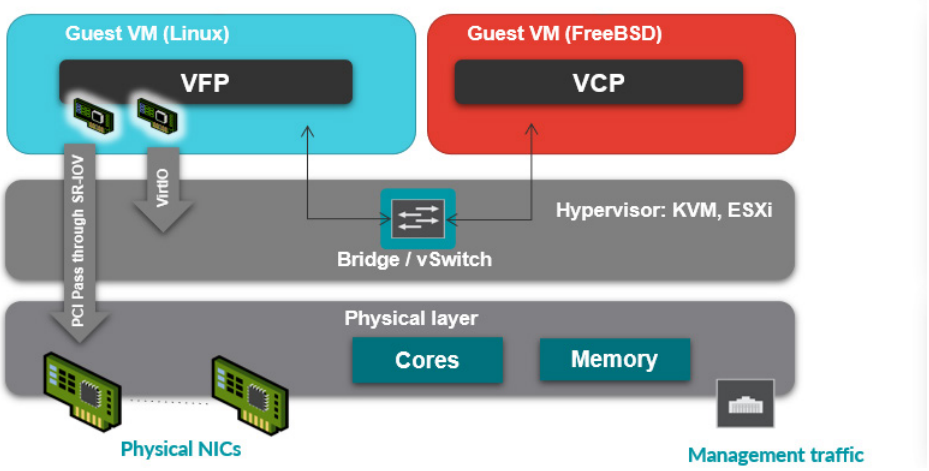

Figure 1.1 *vMX Architecture Overview*

Let's get familiar with the terms used in the architecture in Figure 1.1.

Table 1.1 *vMX Architecture Components*

Components	Description
Physical X86	The server at the hardware layer contains physical network interface cards (NICs), CPUs, and memory. This can be any industry standard x86 server (running Intel processors) that supports virtualization capabilities.
Hypervisors	Over the hardware layer, kernel-based virtual machine (KVM) or VMware ESXi provide the host environment for vMX to run as a VM. This manages the boot process, CPU, memory storage, and various other hardware components of the host.
Guest OS	Junos OS runs as guest OS; it runs the control plane as virtualized routing engine (RE) and the data plane as virtualized packet forwarding engine (PFE/VFP). The PFE does also utilize DPDK for higher performance.
VFP	Virtual machines consist of a Virtual Forwarding Plane (VFP) and one for the Virtual Control Plane (VCP). The VFP VM runs the virtual Trio forwarding plane software.
VCP	Virtual Control Plane. The VCP VM runs Junos OS.
Internal bridge	An internal bridge that is local to the server for each vMX instance enables the communication between VCP and VFP.
vCPU	Represent the logical CPU virtualized by the Intel x86 64-bit CPU. vMX uses one virtual CPU (vCPU) for the RE and at least three vCPU for the VFP.

To route traffic on the vMX, each virtual NIC on the VFP is mapped to a physical NIC, a Linux Bridge, or a VMware vSwitch, based on your configuration. These VFP interfaces are then configured via Junos on the VCP.

As you will see during the labs in this book, you are not required to map a physical NIC to the VFP NIC on the vMX. You can build lab topologies that consist of many routers without having to use a physical NIC anywhere in those topologies. And of course, your more complex topologies can make use of physical NICs and bridges and vSwitches at the same time. Finally, the physical server contains the physical NICs, CPUs, and memory, and provides the management of a vMX instance via a serial console and an Ethernet management interface.

Virtual Control Plane (VCP)

The VCP consists of the Junos OS hosted within a virtual machine. As such, all the usual capabilities you are used to seeing on Junos software are available on the vMX. As Junos is based on FreeBSD, the VCP VM is actually running FreeBSD. The VCP is analogous to the RE in the physical MX.

Virtual Forwarding Plane (VFP)

The VFP consists of a virtualized Trio forwarding plane running on Windriver Linux and is analogous to the FPC/PFE in the physical MX. The VFP makes use of the Intel DPDK libraries to optimize user space packet processing. For more information on DPDK see http://www.intel.com/content/www/us/en/intelligent-systems/intel-tech- nology/packet-processing-is-enhanced-with-software-from-intel-dpdk.html.

The DPDK is designed for fast packet processing and low latency. For the lab, or for low throughputs of up to 100Mbps, a lite mode is available. For high-throughput, a performance flow caching mode is available. In the vMX package, there is one VFP image supplied, and the lite and performance modes are set within the Junos configuration on the VCP.

MORE? For information on enabling performance mode or lite mode, see https://www.juniper.net/documentation/us/en/software/vmx/vmx-getting-started/topics/task/vmx-chassis-flow-caching-enabling.html.

CAUTION If you are using the performance mode VFP, the CPU cores allocated to vMX interfaces will poll constantly (expect to see 100% usage); for this reason you should use the lite version in your lab.

There are three important items to note at this time:

- At the time of this writing, the vMX supports one instance of the VCP, although there is work in progress for vMX to support VCP redundancy. The current release of vMX assumes VCP and VFP are installed on the same physical server, although the architecture does allow for VCP and VFP to be installed on different physical servers.
- There are three components to the software forwarding plane: IO thread, receive thread and transmit thread, and a worker. The worker performs lookups and tasks associated with packet processing and functionality that would normally be found in the Trio ASIC on the physical MX router. The DPDK applies to the receive and transmit threads (the receive thread moves packets from the NIC to the VFP and performs any pre-classification that may be required, and the transmit thread moves packets from the worker to the physical NIC and includes a QoS scheduler to prioritize packets across six queues before being sent to the NIC).
- On Linux, the VMs are managed by an orchestration script provided by Juniper that is used to create, stop, and start the vMX instances. A simple configuration file defines parameters such as memory and vCPUs to allocate to the VCP and VFP. It's not mandatory to use the orchestration script, but doing so creates all the necessary VM configurations for you and provides an easy-to-use mechanism for managing the vMX.

vMX Virtual Machine Connectivity

Clearly the VFP VM and the VCP VM need to be able to communicate directly, so an internal bridge local to the server is required for each vMX instance for this communication.

An external bridge is also required. It enables the management interface on the physical host to be used as the virtual management interface for both VCP and VFP. You will need to configure unique IP and MAC addresses for both the VCP and VFP. The external bridge can be shared by multiple instances of vMX.

This connectivity is shown in Figure 1.2.

On the VCP the internal em1 interface is placed within a routing-instance named _ _ juniper _ private1 _ _ however, this routing-instance and em1 interface are not shown in the configuration. Find more on this in Chapter Five, *Troubleshooting*.

As the Linux KVM release of vMX is managed by an orchestration script, when vMX starts up this script automatically creates the two Linux bridges. On VMware, vSwitches must be created to achieve the same result.

MORE? For a deep dive on the architecture of vMX, see the book, *The MX Series*, 2nd Edition, July 2016, from O'Reilly Media: http://shop.oreilly.com/product/0636920042709.do.

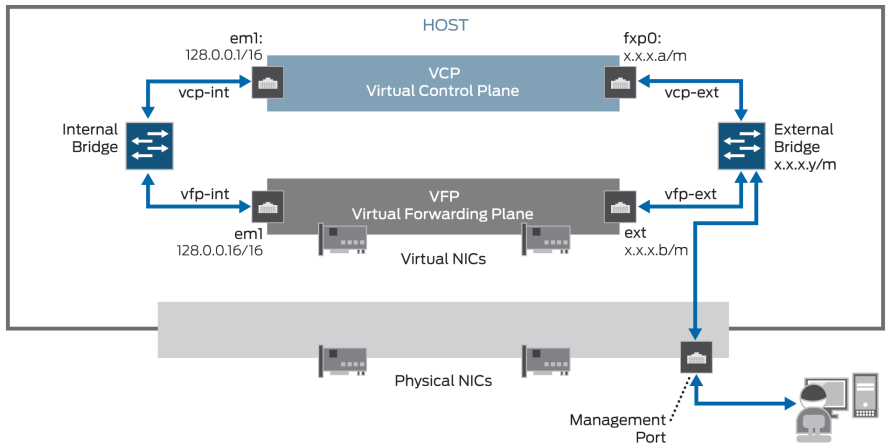

Figure 1.2 vMX Virtual Machine Communication

Data Interfaces and Performance

The vMX router supports the SR-IOV capable NICs, and ports can be renamed to the following Junos OS interface prefix from the default "ge" prefix.

- Gigabit Ethernet (ge)
- 10-Gigabit Ethernet (xe)
- 100-Gigabit Ethernet (et)

For data interfaces, there are a couple of techniques available for packet I/O depending on the required vMX throughput. Both of these techniques are designed to address inefficiencies when fully emulating the physical host.

Which one you choose ultimately depends on your use case for the vMX. Let's get through the specs first and then you can choose your lab setup.

Paravirtualization

For lab use and for throughputs of up to 3Gbps you can use paravirtualization using KVM's virtio drivers or on VMware ESXi by configuring VMXNET3. This paravirtualization technique reduces overhead – essentially the network driver running in the guest virtual machine is aware of the virtual environment and interacts with the hypervisor to execute many functions.

PCI Passthrough with SR-IOV

For high performance use cases, at throughputs of 3Gbps or greater, PCI passthrough with single root I/O virtualization (SR-IOV) is required. Essentially SR-IOV enables the NIC to connect directly to the vMX. As data bypasses the hypervisor I/O performance increases because drivers in the VM directly access the PCI device.

NOTE SR-IOV is fully supported on vMX on KVM and on vMX running on VMware ESXi.

Hardware and Software Requirements

Before you start installing and configuring the vMX, make sure the Virtual Machine (VM) host meets the following recommended hardware, server platform, and software requirements as provided in Table 1.2 and Table 1.3.

Table 1.2 Hardware and Software System Requirements

Requirements	Description
Linux KVM Hypervisor Support	Ubuntu 18.04.3 LTS with Linux 4.15.0-70-generic
Disk Space	Each vMX instance requires 44 GB of disk storage (40 GB for VCP and 4 GB for VFP)
Memory	3 – 18 GB
vCPUs	4 – 9 vCPUs
Network Interface Cards	• SR-IOVIntel X520 NICs using IXGBE driver. • Intel X710 and XL710 NICs with 10G ports using i40e driver. • Intel XL710Q-DA2 NIC with 40G ports using i40e driver.
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.
Sample System Configuration	• For lab simulation and low performance (less than 100 Mbps) use cases, any x86 processor (Intel or AMD) with VT-d capability. • For all other use cases, Intel Ivy Bridge processors or later are required.

NOTE For the latest updates and additions of new supported flavors, please confirm these tables with current documentation: https://www.juniper.net/documentation/en_US/vmx/topics/reference/general/vmx-hw-sw-minimums.html.

Table 1.3 System Requirements for Lite Mode and Performance Mode

Requirement	Lite Mode (Lab Simulation)	Performance Mode (Normal Operations)
Number of cores	Minimum of 4 1 for VCP 3 for VFP	Minimum of 9* 1 for VCP 8 for VFP
Memory	Minimum of 3 GB 1 GB for VCP 2 GB for VFP	Minimum of 5 GB 1 GB for VCP 4 GB for VFP Recommended of 16 GB 4 GB for VCP 12 GB for VFP Additional 2 GB recommended for host OS

*For typical performance mode configurations, we recommend the following formula to calculate the minimum vCPUs required by the VFP: Without QoS—(4 * number-of-ports) + 4; With QoS—(5 * number-of-ports) + 4.

Licensing

You can download the vMX software BASE application package with 1 Mbps bandwidth and evaluate it without a license. To use additional features, you must order the appropriate license. You can truly start small and *pay-as-you-grow* with the vMX.

Licensing is based on a combination of throughput and features and the lowest available throughput license is 100Mbps.

An application package is associated with a bandwidth license. The vMX provides egress bandwidth in the following capacities: 100 Mbps, 250 Mbps, 500 Mbps, 1 Gbps, 5 Gbps, 10 Gbps, and 40 Gbps.

Bandwidth licenses that are *not* associated with a specific application package apply to all application packages. Bandwidth licenses are cumulative. For example, if you add a 500 Mbps license and a 1 Gbps license, you are entitled to use 1.5 Gbps of capacity.

NOTE If you delete all valid licenses, you can only use the BASE application package with 1 Mbps bandwidth.

Table 1.4 vMX Licensing Package

Package	Details
Base	IP routing with 256,000 routes in the RIB/FIB. Provides basic Layer 2 functionality, Layer 2 bridging, and switching. Layer 2 features include Layer 2 VPN, VPLS, EVPN, VXLAN, and Layer 2 Circuit.
Advance	All the features in the Base license. IP routing with routes up to 2,000,000 in the RIB/FIB (8 million for 10G or above). Enabled are IP and MPLS switching for unicast and multicast applications. 16 instances of Layer 3 VPN.
Premium	All the features in the Base and Advance application packages. 4,000,000 Layer 3 VPN for IP and multicast. Limited to 250 VPN instances (L2 and L3 VPN).

NOTE Starting with Junos OS Release 19.2R1, Juniper Agile Licensing introduces a new capability that significantly improves the ease of license management network wide. See the Juniper Agile Licensing Guide for more details on how to obtain, install ,and use the License Manager.

To use the vMX feature licenses in the Junos OS Release 19.2R1 version, you need new license keys. Previous license keys will continue to be supported for previous Junos OS releases but for the Junos OS 19.2R1 release and later you need to carry out a one-time migration of existing licenses. Contact Juniper Customer Care to exchange previous licenses: https://www.juniper.net/us/en/setup/license/#tab=dtabs-1.

NOTE Juniper also offers several virtualized *hands-on* labs services that are designed to allow you to learn about Juniper's product and solution functionality. You can test the vMX in addition to that of other products such as vSRX, Contrail Enterprise Multicloud, and much more with built topologies. Explore Juniper Networks vLabs here https://vlabs.juniper.net.

MORE? You can explore options to quickly connect with the networking solution you need at https://www.juniper.net/us/en/try/.

You can configure the physical MX Series routers to run in different network services modes.

A network services mode defines how the chassis recognizes and uses certain modules. When you set a physical MX router to enhanced-ip network services mode, only the MPC/MIC modules and MS-DPC modules are powered on in the chassis.

This also means that the network services mode can restrict the available Layer 2, Layer 2.5, and Layer 3 features that are available on the MX chassis. For example, if you configure Enhanced Ethernet mode then certain BGP functions will be restricted and there will be no support for Layer 3 VPNs, which also means that unless you are using Enhanced IP mode there will be limited support for Layer 3 features, although Layer 2.5 features such as VPLS will still be supported.

NOTE An unlicensed vMX instance is locked to a network-services mode of Enhanced Ethernet and this means that only the Layer 2.5 features are available. BGP is available but data plane support applies only to Ethernet and MPLS.

As soon as you apply a license to the vMX (including a trial license) the network services mode is automatically changed to enhanced-IP and all the Layer 2 and Layer 3 features become available up to the limits of the applied license. That is, when you use the vMX software with BASE license, you can run it in Enhanced Ethernet mode. You need to apply for an ADVANCED or PREMIUM license to work in enhanced-IP mode.

MORE? You can find out more about the Enhanced Ethernet mode restrictions here: https://www.juniper.net/documentation/us/en/software/junos/chassis/topics/topic-map/chassis-guide-tm-config-ntwrk-srvics-mode.html#id-feature-restrictions-on-mx-series-routers-running-in-ethernet-network-services-mode-or.

Supported Platforms for vMX

Next, let's review the options available for you to install the vMX virtual router. You can install the vMX on:

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

This *Day One* book uses the Linux virtualization solution, KVM, to spin up the virtual instances of the control and forwarding planes. Multiple instances of the vMX can be run on the same physical hardware, and if needed, other KVM virtual machines can also be running. It is probably no surprise that Juniper vMX uses Linux and KVM, as Linux and KVM are used with many other Juniper products such as the vSRX Series.

Chapter 2

Getting Started

Okay, now that you have some background on the vMX, the fun can begin! This chapter walks you through a complete build of the vMX, starting with the installation and set up of the Ubuntu host OS for vMX.

Once the host OS is ready, with the prerequisite packages installed, you will be able to see how vMX is built and configured – from orchestration scripts to configuration files.

Installing vMX

At the time of writing this book, we used the latest version of the vMX available running, Junos OS 20.2R2. You can download the most recent, up-to-date software package at https://support.juniper.net/support/downloads/?p=vmx /.

Be sure to check for new releases depending on when you are reading these pages.

The Ubuntu version used in this book is Ubuntu 18.04.3 LTS (Linux 4.15.0-135-generic x86_64).

Juniper supports the use of Ubuntu 18.04.3 LTS for the vMX host operating system and the KVM hypervisor for Junos OS Release 20.1R1 onwards. When choosing an Ubuntu release for your hypervisor host, make sure that the Ubuntu version is supported by the preferred vMX release. Check here before doing so: https://www.juniper.net/documentation/us/en/software/vmx/vmx-getting-started/topics/concept/vmx-hw-sw-minimums.html. The installation of vMX on Ubuntu is a straightforward process.

NOTE If you are doing this lab build on a MacBook or PC with Ubuntu running as a VM, allocate at least 50 GB hard drive, 12GB RAM, four vCPUs, and two vNICs (one for management, one for data) to the Ubuntu VM. The VM must also be enabled to support Nested Virtualization within the VM.

After installation of Ubuntu 18.04 on your host, make sure that your OS is ready by verifying software and kernel version and installing any required Linux packages.

Verifying Software and Kernel Version

Use the following steps to discover your host machine configuration.

Step 1: Log in to the Ubuntu host machine that you'll be using for vMX using SSH.

Step 2: Review the host system configuration - such as the name of the host, its software version, the 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:

```
user@host:~# uname
```

Linux:

```
user@host:~# uname -a
Linux ix-
ubuntu-03 4.15.0-135-generic #139-Ubuntu SMP Mon Jan 18 17:38:24 UTC  2021 x86_64 x86_64 x86_64 GNU/
Linux
```

Step 3: Check the Ubuntu version:

```
user@host:~# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.3 LTS"
```

As mentioned, Ubuntu version 18.04.3 is the qualified release and updating all of the installed packages may cause issues. Updating the packages is not a necessary step for vMX, but if you wish to update the packages anyway, on Ubuntu it is done using the APT package manager.

Step 4: Optional if you are using Ubuntu 18.04.3.

- Update the list of available packages and their versions:

```
user@host:~# apt-get update
```

- Install latest versions of the packages you have:

```
user@host:~# apt-get upgrade
```

- Install KVM and other required packages:

```
user@host:~# sudo apt-get install qemu-kvm libvirt-bin bridge-utils
```

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

```
user@host:~# sudo apt-get install virt-manager
```

- Install the QEMU system package:

```
user@host:~# sudo 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-s390x qemu-system-sparc
Suggested packages:
  qemu samba vde2 qemu-efi openbios-ppc openhackware openbios-sparc
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-s390x qemu-system-sparc
0 upgraded, 8 newly installed, 0 to remove and 63 not upgraded.
Need to get 41.1 MB of archives.
After this operation, 219 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
<snip>
```

After a short time your vMX host OS will be updated and ready to use.

Preparing the System for vMX

Now it's time to install the prerequisite packages for vMX.

Upgrading the Kernel (Optional)

Depending on the exact version of Ubuntu server that was installed, you may also need to upgrade the kernel packages. Juniper recommends that you use Linux Kernel 4.15.0-70-generic. You can skip this step if you are using Ubuntu 18.04.

Here are the commands to install the latest Linux Kernel:

```
user@host:~$ sudo apt-get install linux-image-4.15.0-70-generic
user@host:/etc/grub.d$ sudo update-grub
```

Install Prerequisite System Packages

Some of these packages will already have been installed during the Ubuntu install process, but the complete list is provided below. Again, this is done using `apt-get`:

```
user@host:~$ apt-get install bridge-utils qemu-kvm libvirt-bin python python-
netifaces vnc4server libyaml-dev python-yaml numactl libparted0-dev libpciaccess-dev libnuma-
dev libyajl-dev libxml2-dev libglib2.0-dev libnl-3-dev python-pip python-dev libxslt1-dev
```

The prerequisites and any package dependencies will now be installed.

Verifying Libvirt Version (Optional)

Libvirt is 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 vMX instances.

Ubuntu 18.04.3 supports libvirt version is 4.0.0. Upgrading libvirt in Ubuntu 18.04 is not required.

NOTE It's recommended that you skip the libvirt upgrade if you are building vMX for lab purposes, or if you plan to run the virtual forwarding plane in Lite mode.

Check the installed version of libvirt:

```
user@host:~# virsh version
Compiled against library: libvirt 4.0.0
Using library: libvirt 4.0.0
Using API: QEMU 4.0.0
Running hypervisor: QEMU 2.11.1
```

All looks good. Now the Ubuntu host is ready for vMX and you can move on with the installation and configuration of the vMX itself.

Installing and Configuring vMX

For this lab-based build, you should use virtio for the virtual NIC. As mentioned earlier, there are two modes of VFP operation: a lite mode PFE for labs and a performance mode for normal operation. You should use the lite mode, which is the default configuration. Let's get started!

Download the vMX from: https://support.juniper.net/support/downloads/?p=vmx.

```
user@host:~# wget -O vmx-bundle-20.2R2.11.tgz "https://cdn.juniper.net/software/vmx/20.2R2.11/
vmx-bundle-20.2R2.11.tgz?SM_USER=name&__gda__=1612408693_69c642b57c6095612d61be1666e4d997"
--2021-02-04 05:38:41--  https://cdn.juniper.net/software/vmx/20.2R2.11/vmx-b undle-20.2R2.11.
tgz?SM_USER=user1__gda__=1612417995_8aec0cb99636e6170ef61 458eeb048b5
Resolving cdn.juniper.net (cdn.juniper.net)... 23.203.176.210
Connecting to cdn.juniper.net (cdn.juniper.net)|23.203.176.210|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1268426361 (1.2G) [application/octet-stream]
Saving to: âvmx-bundle-20.2R2.11.tgzâ

vmx-bundle-20.2R2.1 100%[=================>]   1.18G  3.31MB/s    in 4m 48s

2021-02-04 05:43:31 (4.20 MB/s) - âvmx-bundle-20.2R2.11.tgzâ saved [126842636 1/1268426361]
```

Let's have a look at the vMX package contents:

```
user@host:~$ ls
vmx-bundle-20.2R2.11.tgz
```

Extract the package in your home directory:

```
user@host:~$ tar xzvf vmx-bundle-20.2R2.11.tgz
ou
```

Check the vMX installed package contents:

```
user@host:~# ls
vmx  vmx-bundle-20.2R2.11.tgz
```

Check vMX images located within in the "images" directory:

```
user@host:~# ls vmx/images/
junos-vmx-x86-64-20.2R2.11.qcow2  metadata-usb-fpc1.img  metadata-usb-fpc5.img  metadata-usb-fpc9.
img  metadata-usb-service-pic-10g.img  vmxhdd.img
metadata-usb-fpc0.img             metadata-usb-fpc2.img  metadata-usb-fpc6.img  metadata-usb-re0.
img   metadata-usb-service-pic-2g.img
metadata-usb-fpc10.img            metadata-usb-fpc3.img  metadata-usb-fpc7.img  metadata-usb-re1.
img   metadata-usb-service-pic-4g.img
metadata-usb-fpc11.img            metadata-usb-fpc4.img  metadata-usb-fpc8.img  metadata-usb-re.
img    vFPC-20201014.img
```

Write down the following image details, as you might need them while installing the package:

- VCP image - junos-vmx-x86-64-20.2R2.11.qcow2
- VFP image - vFPC-20201014.img
- Routing Engine - vmxhdd.img

Check the configuration files. The configuration files are located within the config directory. The main config for vMX is defined in vmx.conf, and the configuration for vMX interfaces (virtio) is within vmx-junosdev.conf:

```
user@host:~# cd vmx/config
user@host:~/vmx/config# ls
samples  vmx.conf  vmx-junosdev.conf
```

Now let's examine the configuration files vmx.conf and vmx-junosdev.conf.

Configuring and Deploying a Single Instance of vMX on KVM

To begin, you need to set up the vMX configuration file. By default, this is done by editing config/ vmx.conf, however, you can create your own configuration file and use the script `--cfg` option to specify it. The configuration file uses the YAML format.

NOTE Multiple instances of vMX can run on the same physical host. You simply need to define additional configuration files.

Host Configuration

Display the configuration file:

```
user@host:~/vmx/config# cat vmx.conf ################################################################
#
#  vmx.conf
#  Config file for vmx on the hypervisor.
#  Uses YAML syntax.
#  Leave a space after ":" to specify the parameter value.
#
############################################################

---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier                : vmx1   # Maximum 6 characters
    host-management-interface : eth0
    routing-engine-image      : "/home/vmx/vmxlite/images/jinstall64-vmx.img"
    routing-engine-hdd        : "/home/vmx/vmxlite/images/vmxhdd.img"
    forwarding-engine-image   : "/home/vmx/vmxlite/images/vPFE.img"

---
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                 # Max 10 characters

---
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 1024
    console_port: 8601

    interfaces  :
      - type      : static
        ipaddr    : 10.102.144.94
        macaddr   : "0A:00:DD:C0:DE:0E"

---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 4096
    vcpus       : 4
    console_port: 8602
    device-type : virtio

    interfaces  :
      - type      : static
        ipaddr    : 10.102.144.98
        macaddr   : "0A:00:DD:C0:DE:10"

---
#Interfaces
JUNOS_DEVICES:
   - interface           : ge-0/0/0
     mac-address         : "02:06:0A:0E:FF:F0"
     description         : "ge-0/0/0 interface"
```

```
   - interface            : ge-0/0/1
     mac-address          : "02:06:0A:0E:FF:F1"
     description          : "ge-0/0/0 interface"

   - interface            : ge-0/0/2
     mac-address          : "02:06:0A:0E:FF:F2"
     description          : "ge-0/0/0 interface"

   - interface            : ge-0/0/3
     mac-address          : "02:06:0A:0E:FF:F3"
     description          : "ge-0/0/0 interface"
```

Display the vMX interfaces configuration file:

```
user@host:~/vmx/config# cat vmx-junosdev.conf
################################################################
#
#  vmx-junos-dev.conf
#  - Config file for junos device bindings.
#  - Uses YAML syntax.
#  - Leave a space after ":" to specify the parameter value.
#  - For physical NIC, set the 'type' as 'host_dev'
#  - For junos devices, set the 'type' as 'junos_dev' and
#    set the mandatory parameter 'vm-name' to the name of
#    the vPFE where the device exists
#  - For bridge devices, set the 'type' as 'bridge_dev'
#
################################################################
interfaces :

     - link_name  : vmx_link1
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : bridge_dev
           dev_name    : bridge1

     - link_name  : vmx_link2
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx2
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : bridge_dev
           dev_name    : bridge1

     - link_name  : vmx_link3
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/1
       endpoint_2 :
         - type        : host_dev
           dev_name    : eth3
```

```
  - link_name  : vmx_link4
    endpoint_1 :
      - type        : junos_dev
        vm_name     : vmx1
        dev_name    : ge-0/0/2
    endpoint_2 :
      - type        : junos_dev
        vm_name     : vmx2
        dev_name    : ge-0/0/2
```

If you need to make small edits to the vmx.conf file:

- First set an instance identifier for the vMX instance, here it is set to vmx1.
- Next update the configuration file to reflect the absolute path to the image files.

Table 2.1 Changes to vmx.conf File

Image Files	Changes
host-management-interface	Set management interface on the physical host. This interface will be bridged to the vMX instance's own management interfaces on the VCP and VFP.
routing-engine-image	qcow2 file location
routing-engine-hdd	vmxhdd.img location
forwarding-engine-image	vFPC.img file location

We made the following changes in this example, using the root directory since we are the root user:

```
HOST:
    HOST:
    identifier                : vmx1   # Maximum 6 characters
    host-management-interface : ens160
    routing-engine-image      : "/root/vmx/images/junos-vmx-x86-64-20.2R2.11.qcow2"
    routing-engine-hdd        : "/root/vmx/images/vmxhdd.img"
    forwarding-engine-image   : "/root/vmx/images/vFPC-20201014.img"
```

VCP and VFP Configuration

Now let's configure the parameters for the control and forwarding planes. These are also defined in the vmx.conf configuration file.

For vMX 20.2, allocate the following (minimum requirement):

- VCP: 1 vCPU and 1 GB RAM
- VFP: 3 vCPUs and 2 GB RAM

NOTE The labs in this book are running Ubuntu as a nested VM, with 4 GB allocated to the forwarding plane. Depending on the features and version of vMX that you are using, 4 GB should be fine for your lab purposes. 1 GB should be the absolute minimum on the control plane. *Please don't do this in a production environment because it is not a Juniper supported configuration and if something goes wrong, JTAC won't help you!*

NOTE The VFP device type is set to virtio for the interfaces.

Next you can see that a bridge is also defined – this is the management interface bridge mentioned earlier. You need to set an IP address for the control plane and forwarding plane – make sure to use an IP on the same subnet as the host management network:

```
---
#External bridge configuration BRIDGES:
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                # Max 10 characters

---
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 8601

    interfaces  :
      - type      : static
        ipaddr    : 10.1.1.2
        macaddr   : "0A:00:DD:C0:DE:0E"

---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 4096
    vcpus       : 4
    console_port: 8602
    device-type : virtio

    interfaces  :
      - type      : static
        ipaddr    : 10.1.1.3
        macaddr   : "0A:00:DD:C0:DE:10"
```

The default MAC addresses used in the configuration file are taken from the locally administered MAC address ranges. For the time being, you have only a single instance of vMX running, so no other VCP or VFP parameters need to be changed at this point.

Interface Configuration (virtio)

Now let's configure the interface for the vMX. You will only be using one interface in this lab setup but many more can be configured. Just comment out the other interfaces, leaving only ge-0/0/0 defined:

```
---
---
#Interfaces
JUNOS_DEVICES:
   - interface            : ge-0/0/0
     mac-address          : "02:06:0A:0E:FF:F0"
     description          : "ge-0/0/0 interface"
```

NOTE Things are done slightly differently for an SR-IOV configuration as there are a few additional parameters to configure. SR-IOV is out of scope for this lab but not too terribly difficult. Try it yourself! There are sample configuration files for both virtio and SR-IOV in the vMX package directory config/ samples.

You also need to create Linux bridges to link vMX ge-0/0/0 to an interface on the physical host. By default, this is done in the device binding configuration file, config/vmx-junosdev.conf, however, you can create your own configuration file and use the `--cfg` option to specify it. The vMX orchestration scripts do all the heavy lifting for you to set up the bridges.

The device binding file uses YAML, enabling a flexible configuration for connecting VFP endpoints to a physical NIC, another vMX instance, or to another Linux bridge.

The parameters in the configuration are:

- link-name: This is the name of the Linux bridge; it can be up to 15 characters long and must be unique.
- mtu: The default is 1500 but can be increased to 9500.
- endpoint: This can be a vMX instance (junos _ dev), a host interface (host _ dev), or a bridge (bridge _ dev). For endpoint type junos _ dev. The setting vm _ name represents the actual name of the vMX instance.
- dev _ name: Represents the interface name or bridge name.

You need to create a new Linux bridge between host interface eth1 and ge-0/0/0 on vmx1. Modify the configuration file config/vmx-junosdev.conf like this:

```
user@host:~/vmx/config# cat vmx-junosdev.conf
################################################################
#
#  vmx-junos-dev.conf
#  - Config file for junos device bindings.
#  - Uses YAML syntax.
#  - Leave a space after ":" to specify the parameter value.
```

```
#  - For physical NIC, set the 'type' as 'host_dev'
#  - For junos devices, set the 'type' as 'junos_dev' and
#    set the mandatory parameter 'vm-name' to the name of
#    the vPFE where the device exists
#  - For bridge devices, set the 'type' as 'bridge_dev'
#
##############################################################
interfaces :

     - link_name  : vmx_link1
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : bridge_dev
           dev_name    : bridge1
```

You can see that the lab has defined a single Linux bridge named vmx _ link1 and it will use this bridge to link ge-0/0/0 on the instance vmx1 to the host physical interface bridge1.

You will need to use the vMX orchestration script to activate this binding and create the Linux bridge but let's do that once you have successfully deployed the vMX instance.

Deploying Your Instance of vMX

Now that the vMX has been configured, it's time for you to deploy your instance. This is done using the orchestration script. Your vMX instance will be created and automatically started by the script.

Make sure that you specify the `-lv` parameter for verbose logging because this is really going to help you with troubleshooting if the scripts run into a problem. Once corrected, the installer completes and starts up the vMX – remember there are two VMs that must be started, the VCP and the VFP:

```
user@host:~/vmx# ./vmx.sh -lv --install
================================================
    Welcome to VMX
================================================
Date.....................................08/10/20 09:15:07
VMX Identifier...........................vmx1
Config file................................/root/vmx/config/vmx.conf
Build Directory............................/root/vmx/build/vmx1
Assuming kvm hypervisor.....................
Virtualization type.....................kvm
Junos Device type.......................virtio
Environment file............................/root/vmx/env/ubuntu_virtio.env
Junos Device Type.......................virtio
Initialize scripts......................[OK]
[OK]
[OK]
================================================
    VMX Environment Setup Completed
================================================
```

```
================================================
    VMX Install & Start
================================================
Linux distribution.............................ubuntu
Check GRUB.....................................[Disabled]
Installation status of qemu-kvm................[OK]
Installation status of libvirt-bin.............[OK]
Installation status of bridge-utils............[OK]
Installation status of python..................[OK]
Installation status of libyaml-dev.............[OK]
Installation status of python-yaml.............[OK]
Installation status of numactl.................[OK]
Installation status of libnuma-dev.............[OK]
Installation status of libparted0-dev..........[OK]
Installation status of libpciaccess-dev........[OK]
Installation status of libyajl-dev.............[OK]
Installation status of libxml2-dev.............[OK]
Installation status of libglib2.0-dev..........[OK]
Installation status of libnl-dev...............[OK]
Check Kernel Version...........................[Disabled]
Check Qemu Version.............................[Disabled]
Check libvirt Version..........................[Disabled]
Check virsh connectivity.......................[OK]
[OK]
[OK]
================================================
    Pre-Install Checks Completed
================================================
Check RE state.................................[Not Running]
[OK]
Check for VM vfp-vmx1..........................[Not Running]
[OK]
Check if bridge br-ext exists..................[Yes]
Get Configured Management Interface............ens160
Find existing management gateway...............br-ext
Mgmt interface needs reconfiguration...........[Yes]
Gateway interface needs change.................[Yes]
Get Management Address and Mask................
Check if br-ext has valid IP address and mask.....[Yes]
Get Management Gateway.........................10.102.70.254
Del ens160 from br-ext.........................[OK]
Configure ens160...............................[Yes]
Cleanup VM bridge br-ext.......................[OK]
Cleanup VM bridge br-int-vmx1..................[OK]
Cleanup VM bridge br-fab-vmx1..................[OK]
================================================
    VMX Stop Completed
================================================
Check VCP image................................[OK]
Check VFP image................................[OK]
Check VCP Config image.........................[OK]
Check management interface.....................[OK]
Setup huge pages to 4096.......................[Already configured]
[OK]
Attempt to kill libvirtd.......................[OK]
Attempt to start libvirt-bin...................[OK]
Sleep 2 secs...................................[OK]
Check libvirt support for hugepages............[OK]
================================================
    System Setup Completed
```

```
================================================
Get Management Address of ens160.................[OK]
Generate libvirt files...........................[OK]
Sleep 2 secs.....................................[OK]
Find configured management interface.............ens160
Find existing management gateway.................ens160
Check if ens160 is already enslaved to br-ext.....[No]
Gateway interface needs change...................[Yes]
Create br-ext....................................[OK]
Get Management Gateway...........................10.102.70.254
Flush ens160.....................................[OK]
Start br-ext.....................................[OK]
Bind ens160 to br-ext............................[OK]
Get Management MAC...............................00:50:56:93:fb:99
Assign Management MAC 00:50:56:93:fb:99..........[OK]
Add default gw 10.102.70.254.....................[OK]
Create br-int-vmx1...............................[OK]
Start br-int-vmx1................................[OK]
[OK]
Define vcp-vmx1..................................[OK]
Start vcp-vmx1...................................[OK]
Define vfp-vmx1..................................[OK]
Wait 2 secs......................................[OK]
Start vfp-vmx1...................................[OK]
Wait 2 secs......................................[OK]
================================================
    VMX Bringup Completed
================================================
Check if br-ext is created.......................[Created]
Check if br-int-vmx1 is created..................[Created]
Check if VM vcp-vmx1 is running..................[Running]
Check if VM vfp-vmx1 is running..................[Running]
Check if tap interface vfp-ext-vmx1 exists.......[OK]
Check if tap interface vfp-int-vmx1 exists.......[OK]
Check if tap interface vcp-ext-vmx1 exists.......[OK]
Check if tap interface vcp-int-vmx1 exists.......[OK]
================================================
    VMX Status Verification Completed.
================================================
Log file.........................................../root/vmx/build/vmx1/logs/vmx_1597076107.log
================================================
    Thank you for using VMX
================================================
```

If anything goes wrong the installer will abort and you will be given an error message as shown here:

```
user@host:~/vmx# ./vmx.sh -lv --install ================================================
Welcome to VMX
================================================ Date.................02/23/20 12:59:00
VMX Identifier mx1
Config file. /root/vmx/config/vmx.
conf
Build ……..
Start vfp-vmx1 [Failed]
error: Failed to start domain vfp-vmx1
error: internal error: early end of file from monitor: possible problem: file_ram_
```

```
alloc: can't mmap RAM pages: Cannot allocate memory
Log file.....................................
/root/vmx/build/vmx1/logs/vmx_1456232340.log
================================================
Aborted!. 1 error(s) and 0 warning(s)
================================================
```

In this case VFP isn't starting because the Ubuntu host does not have enough memory assigned. Because this is the lab and we're running the Ubuntu KVM server itself as a VM, it's a quick fix to assign some more memory to it.

Now let's take a quick look at what the orchestration script has done to deploy this vMX instance. All the images and settings for a particular vMX instance are located within the build/ directory. You can see that for the instance vMX1 there are three directories: images, logs, and xml:

```
user@host:~/vmx/build/vmx1# ls
images  logs  xml
```

The images subdirectory is where the software image files are located for the vMX instance. When you deploy a vMX instance, the orchestration script will copy the package image files to this vMX instance-specific location:

```
user@host:~/vmx/build/vmx1/images# ls
junos-vmx-x86-64-20.2R2.11.qcow2  metadata-usb-fpc0.img  metadata-usb-re.img  vFPC-20201014.
img vmxhdd.img
```

This also enables you to have multiple vMXs on the same system, each running different versions of the Junos OS. The image file vmxhdd.img is used by the VCP to store configuration information.

The logs directory is where the orchestration scripts place the log files. This is a good place to look if you have any problems managing your vMX deployment or during a stop/start operation.

The XML directory is where copies of the libvirt XML files are stored. These XML files contain the configuration data for the Internal/External bridges and the VCP/VFP virtual machines. Later in this chapter there is more on how librvirt uses these configuration files to start up the vMX.

You might also be interested in knowing how much disk space an instance of vMX will require. It's around 2.1G – this is because all of the image files are copied to the vMX instance-specific build area:

```
user@host:~/vmx/build# du -sh vmx1
3.9G    vmx1
```

Linux Bridges and Managing a Virtio Binding

At this point the vMX is running and since you already configured the binding when you edited the config/ vmx-junosdev.conf file, all that remains to be done is to activate the configuration. But first let's review what Linux bridges the vMX script just created when the vMX instance was deployed. This is done using the shell `brctl show` command:

```
user@host:~/vmx/build# brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.00505693fb99       yes             br-ext-nic
                                                        ens160
                                                        vcp-ext-vmx1
                                                        vfp-ext-vmx1
br-int-vmx1     8000.5254002b5b25       yes             br-int-vmx1-nic
                                                        vcp-int-vmx1
                                                        vfp-int-vmx1
virbr0          8000.5254001bd668       yes             ge-0.0.0-vmx1
                                                        ge-0.0.1-vmx1
                                                        ge-0.0.2-vmx1
                                                        ge-0.0.3-vmx1
                                                        virbr0-nic
```

You can see the bridges in the output that the vMX automatically creates when started.

Bridge `br-ext` is the external bridge that is used for management of the vMX and the KVM host. You can see that `ens160` on the physical host and the management interfaces on the VCP and VFP have been added to this bridge, which can be shared by multiple vMX instances.

Bridge `br-int-vmx1` is the internal bridge used for communication between the VCP and VFP that together make up a particular vMX instance. You can see here that the internal interfaces on the VCP and VFP have been added to this bridge. Separate internal bridges are required per vMX instance, which is why this one is named with the "-vmx1" suffix.

Now it's time to activate the virtio binding. First check that it has not already been activated. Again, you will be using the orchestration script that Juniper provides with the vMX:

```
user@host:~/vmx# sudo ./vmx.sh --bind-check
Checking package ethtool.........................[OK]
Check Bridge port bridge1(ge-0.0.0-vmx1).........[ Not Present]
```

Well, from the output it is pretty clear that the binding is missing. This time let's use the `bind-dev` option to create the binding:

```
user@host:~/vmx# sudo ./vmx.sh --bind-dev
Checking package ethtool.........................[OK]
Bind Bridge port bridge1(ge-0.0.0-vmx1)..........[OK]
```

You might encounter an error when binding bridges:

```
user@host:~/vmx# sudo ./vmx.sh --bind-dev
Checking package ethtool.........................[OK]
Bind Bridge port bridge1(ge-0.0.0-vmx1)..........[OK]
Numa node for eth1..............................-1
Cores servicing numa node -1.....................
Pid of vfp-vmx1. 20804
Pin vhost-20804 (PID=20807) to cores taskset: failed to parse CPU list:
[Failed]
Pin vhost-20804 (PID=20806) to cores taskset: failed to parse CPU list:
[Failed]
```

The `taskset` command is used to achieve better performance in virtio mode, however, the error can be ignored for the purposes of your lab so long as the bindings are present.

Try It Yourself *Connect the vMX to a KVM Host Interface and monitor traffic with tcpdump.*

Bind a host interface with the configuration as demonstrated in this chapter and test to see if you can send traffic from the vMX via the physical interface. Use the `tcpdump` option on the KVM host interface to monitor traffic being bridged between the vMX and the host interface.

Modify the configuration if you wish, and then apply and check the new binding again. This configuration will bind ge-0-0/0 to bridge1 (adapt to your environment if necessary):

```
- link_name  : vmx_link1
  mtu        : 1500
  endpoint_1 :
    - type        : junos_dev
      vm_name     : vmx1
      dev_name    : ge-0/0/0
  endpoint_2 :
    - type        : bridge_dev
      dev_name    : bridge1
```

Connect to the vMX Instances

You can now connect to the vMX via the serial console. This is done using the vmx.sh script again.

Serial Console

You will need to specify `vcp` (control plane) or `vfp` (forwarding plane), as well as the instance name as options. In the example below, a console connection is being made to the VCP on the instance named `vmx1`:

```
user@host:~/vmx# ./vmx.sh --console vcp vmx1
--
```

```
Login Console Port For vcp-vmx1 - 8601
Press Ctrl-] to exit anytime
--
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

r
FreeBSD/amd64 (Amnesiac) (ttyu0)

login: root
```

TIP To break out of a console connection to the vMX use the standard Ctrl-] escape keyboard sequence. The default login credentials for the VCP are root, no password, and for the VFP root, root.

TIP If you are new to the Junos CLI, see *Day One: Exploring the Junos CLI, Second Edition* at https://www.juniper.net/dayone.

SSH

Remember that the virtual management interface on the VCP (interface fxp0) is bridged to the physical host management interface and multiple instances of vMX are able to share this external bridge.

This means that you can also use SSH to access the Junos OS on the vMX. You will find that using SSH to configure vMX makes things a lot easier for your lab build. It's done like this.

Console in to the vMX instance and set an IP address on the management interface. As the physical host's management interface is bridged to the VCP management interface, use an IP address from the same subnet as the physical host's management IP:

```
set interfaces fxp0 unit 0 family inet address 10.1.1.2/24
```

Then enable the SSH service. Here the root login is enabled but you don't want to do that outside of a lab:

```
set system services ssh
set system services ssh allow-root-login
```

Set the hostname and a password for the root user, if you have not done so already:

```
set system host-name vmx1
set system root-authentication plain-text-password
```

Now commit the configuration and exit the console session. You should be able to SSH directly to the vMX using the IP address that was just configured on the fxp0 interface:

```
user@host:~/vmx# ssh root@10.1.1.2
The authenticity of host '10.1.1.2 (10.1.1.2)' can't be established.
ECDSA key fingerprint is SHA256:altUKXvSoLSqTKipbRUPgGWC1/6YP6wwPCJ8d2e17CU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.1.1.2' (ECDSA) to the list of known hosts.
Password:
--- JUNOS 20.2R2.11 Kernel 64-bit  JNPR-11.0-20200608.0016468_buil
root@vmx1:~ #
```

NOTE By default, DHCP is enabled on the br-ext interface and it assigns IP address to the VFP management interface. You can also choose to configure the IP address manually.

Similarly, for the VCP management interface fxp0, you can configure to receive IP address from DHCP or configure the IP address manually.

Managing Licenses

You have now installed Ubuntu and deployed an instance of vMX, but before you can do anything else you need to apply a license to the vMX. This is done via the virtual control plane on Junos.

You can download the vMX software BASE application package with 1 Mbps bandwidth and evaluate it without a license. To use additional features, you must order the appropriate license. See http://www.juniper.net/us/en/dm/free-vmx-trial/ for details.

Adding a License to the vMX

1. Connect to the vMX, log in as root, and start the Junos CLI:

```
root@% cli root>
```

2. Copy the license to the vMX and add the key file by specifying a file name, or do it directly by pasting the key into the terminal as shown here:

```
root> request system license add <filename>
```

Or:

```
root> request system license add terminal
```

3. Verify that the license has been installed correctly:

```
root> show system license

License usage:
                            Licenses     Licenses    Licenses    Expiry
  Feature name                   used    installed      needed
  scale-subscriber                  0         1000           0    permanent
  scale-l2tp                        0         1000           0    permanent
  scale-mobile-ip                   0         1000           0    permanent
```

```
  VMX-BANDWIDTH                  40000        40000           0    permanent
  VMX-SCALE                          3            3           0    permanent

Licenses installed:
  License identifier: JUNOS640113
  License version: 4
  Software Serial Number: XXXXXXX
  Customer ID: vMX-Juniper
  Features:
    vmx-bandwidth-40g - vmx-bandwidth-40g
      permanent
    vmx-feature-premium - vmx-feature-premium
      permanent
```

VMX-BANDWIDTH indicates the licensed bandwidth (in Mbps) and VMX-SCALE indicates the application package. (VMX-SCALE 1 is the BASE package, VMX-SCALE 2 is the ADVANCE package, and VMX-SCALE 3 is the PREMIUM package.) This information is also listed as Features in the Licenses installed section. For example, this output indicates that the 40G perpetual license for the PREMIUM application package is installed.

You can also check the licensing for the PFE:

```
root> show pfe statistics traffic bandwidth

Configured Bandwidth    : 50000000 bps Bandwidth  : 0 bps
Average Bandwidth: 0 bps
```

The vMX is now ready for your lab!

Managing the vMX

Let's run the vMX orchestration script without any options so it will display all available options:

```
user@host:~/vmx# ./vmx.sh

Usage: vmx.sh [CONTROL OPTIONS]
       vmx.sh [LOGGING OPTIONS] [CONTROL OPTIONS]
       vmx.sh [JUNOS-DEV BIND OPTIONS]
       vmx.sh [CONSOLE LOGIN OPTIONS]

    CONTROL OPTIONS:
       --install                    : Install And Start vMX
       --start                      : Start vMX
       --stop                       : Stop vMX
       --restart                    : Restart vMX
       --status                     : Check Status Of vMX
       --cleanup                    : Stop vMX And Cleanup Build Files
       --cfg <file>                 : Override With The Specified vmx.conf File
       --env <file>                 : Override With The Specified Environment .env File
       --build <directory>          : Override With The Specified Directory for Temporary Files
       --help                       : This Menu
```

```
    LOGGING OPTIONS:
       -l                              : Enable Logging
       -lv                             : Enable Verbose Logging
       -lvf                            : Enable Foreground Verbose Logging

    JUNOS-DEV BIND OPTIONS:
       --bind-dev                      : Bind Junos Devices
       --unbind-dev                    : Unbind Junos Devices
       --bind-check                    : Check Junos Device Bindings
       --cfg <file>                    : Override With The Specified vmx-junosdev.conf File

    CONSOLE LOGIN OPTIONS:
       --console [vcp|vfp] [vmx_id]    : Login to the Console of VCP/VFP

    VFP Image OPTIONS:
       --vfp-info <VFP Image Path>     : Display Information About The Specified vFP image

Copyright(c) Juniper Networks, 2015
```

Use these options with the vmx.sh script to stop, start, restart, verify, and clean up an existing vMX:

- cfg file: Use the specified configuration file. The default file is config/vmx.conf. If you do not specify a startup configuration file with this option, the default file is used.
- cleanup: Stop the vMX and clean up the vMX instance. This option will also remove any Linux bridges.

CAUTION! Be careful with this cleanup option. It will delete all of the Junos configuration for a vMX instance!

- restart: Stop and start a running vMX.
- start: Start the vMX instance.
- status: Verify the status of a deployed vMX.
- stop: Stop vMX without cleaning up build files so that the vMX can be started quickly without setup performed by the `--install` option.

Libvirt

If you're interested in what the vmx.sh script does with libvirt and virsh behind the scenes, let's first take a look at where libvirt stores the configuration files for the vMX virtual machines:

```
user@host:/etc/libvirt/qemu# ls
networks  vcp-vmx1.xml  vfp-vmx1.xml
```

The networks directory is where the Linux bridge configurations are created by virsh, and the two XML files vcp-vmx1.xml and vfp-vmx1.xml, are the actual configuration files for the vMX VMs. If you take a look at one of these files you will see what has been set up. The parameters are fairly self-explanatory.

Next, as you can see, the file should not be edited directly. You can make changes by editing the vMX configuration files and re-running the installer, or by using virsh. You can also view this XML file by using virsh's `virsh dumpxml vcp-vmx1` command:

```
user@host:/etc/libvirt/qemu# sudo cat vcp-vmx1.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 edit vcp-vmx1
or other application using the libvirt API.
-->

<domain type='kvm'>
  <name>vcp-vmx1</name>
  <uuid>25fd69ea-00be-41a3-9ab4-2cc5f68ed8c9</uuid>
  <memory unit='KiB'>2000000</memory>
  <currentMemory unit='KiB'>2000000</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <sysinfo type='smbios'>
    <bios>
      <entry name='vendor'>Juniper</entry>
    </bios>
    <system>
      <entry name='manufacturer'>VMX</entry>
      <entry name='product'>VM-vcp_vmx1-161-re-0</entry>
      <entry name='version'>0.1.0</entry>
    </system>
  </sysinfo>
  <os>
    <type arch='x86_64' machine='pc-0.13'>hvm</type>
    <boot dev='hd'/>
    <smbios mode='sysinfo'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <cpu mode='custom' match='exact'>
    <model fallback='allow'>qemu64</model>
    <topology sockets='1' cores='1' threads='1'/>
    <feature policy='disable' name='svm'/>
  </cpu>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/qemu-system-x86_64</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='directsync'/>
      <source file='/root/vmx/build/vmx1/images/junos-vmx-x86-64-20.2R1.10.qcow2'/>
      <target dev='vda' bus='virtio'/>
```

```
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </disk>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='directsync'/>
      <source file='/root/vmx/build/vmx1/images/vmxhdd.img'/>
      <target dev='vdb' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x08' function='0x0'/>
    </disk>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw' cache='directsync'/>
      <source file='/root/vmx/build/vmx1/images/metadata-usb-re.img'/>
      <target dev='vdc' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x09' function='0x0'/>
    </disk>
    <controller type='usb' index='0' model='none'/>
    <controller type='pci' index='0' model='pci-root'/>
    <interface type='bridge'>
      <mac address='0a:00:dd:c0:de:0e'/>
      <source bridge='br-ext'/>
      <target dev='vcp-ext-vmx1'/>
      <model type='virtio'/>
      <driver name='qemu'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
    <interface type='bridge'>
      <mac address='52:54:00:ca:43:a9'/>
      <source bridge='br-int-vmx1'/>
      <target dev='vcp-int-vmx1'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </interface>
    <serial type='tcp'>
      <source mode='bind' host='127.0.0.1' service='8601'/>
      <protocol type='telnet'/>
      <target port='0'/>
    </serial>
    <console type='tcp'>
      <source mode='bind' host='127.0.0.1' service='8601'/>
      <protocol type='telnet'/>
      <target type='serial' port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <input type='keyboard' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes' listen='127.0.0.1'>
      <listen type='address' address='127.0.0.1'/>
    </graphics>
    <sound model='ac97'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </sound>
    <video>
      <model type='cirrus' vram='16384' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

If you want to double check the XML configuration for one of the bridges, then again, it's done by looking at the XML directly with virsh. For example, to view br-ext:

```
user@host:/etc/libvirt/qemu# virsh net-dumpxml br-ext
<network>
  <name>br-ext</name>
  <uuid>a13a0511-21e1-45a8-b395-1e67bcbfe730</uuid>
  <forward mode='route'/>
  <bridge name='br-ext' stp='on' delay='0'/>
  <mac address='52:54:00:9f:a0:77'/>
  <ip address='10.1.1.1' netmask='255.255.255.0'>
    <dhcp>
      <host mac='0A:00:DD:C0:DE:0E' name='vcp-vmx1' ip='10.1.1.2'/>
      <host mac='0A:00:DD:C0:DE:10' name='vfp-vmx1' ip='10.1.1.3'/>
    </dhcp>
  </ip>
</network>
```

Notice that there is a set of DHCP configurations. This is used to assign the management addresses that you defined in the vMX configuration file. Try consoling in to the VFP and check that everything is working correctly:

```
user@host:~/vmx# sudo ./vmx.sh --console vfp vmx1
--
Login Console Port For vfp-vmx1 - 8602
Press Ctrl-] to exit anytime
--
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.


Wind River Linux 9.0.0.20 qemux86-64 console
```

Check the interface configuration:

```
root@vfp-vmx1:~# ifconfig ext
ext       Link encap:Ethernet  HWaddr 0a:00:dd:c0:de:10
          inet addr:10.1.1.3  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::800:ddff:fec0:de10/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25488 errors:0 dropped:9 overruns:0 frame:0
          TX packets:560 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1326668 (1.2 MiB)  TX bytes:75300 (73.5 KiB)

int       Link encap:Ethernet  HWaddr 52:54:00:0f:8f:37
          inet addr:128.0.0.16  Bcast:128.0.255.255  Mask:255.255.0.0
          inet6 addr: fe80::5054:ff:fe0f:8f37/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1745918 errors:0 dropped:76265 overruns:0 frame:0
          TX packets:1552171 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:104378765 (99.5 MiB)  TX bytes:153262072 (146.1 MiB)
```

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:64 errors:0 dropped:0 overruns:0 frame:0
          TX packets:64 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:7427 (7.2 KiB)  TX bytes:7427 (7.2 KiB)
```

And here you can see the IP address assigned to the `ext` interface is the one specified in the XML configuration for `br-ext`.

Virsh

Now let's take a look at a running an instance of vMX using the CLI tool `virsh`. First take a look at the VMs running in the Linux KVM:

```
user@host:~/vmx# sudo virsh list
 Id    Name                           State
----------------------------------------------------
 1     vcp-vmx1                       running
 2     vfp-vmx1                       running
```

Here you can see the two virtual machines, the VCP and the VFP, and notice they are both running. If you want to get some more information on the running VMs, then use the `dominfo` command:

```
user@host:~/vmx# sudo virsh dominfo vcp-vmx1
Id:             1
Name:           vcp-vmx1
UUID:           25fd69ea-00be-41a3-9ab4-2cc5f68ed8c9
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       1601.5s
Max memory:     2000896 KiB
Used memory:    2000896 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: apparmor
Security DOI:   0
Security label: libvirt-25fd69ea-00be-41a3-9ab4-2cc5f68ed8c9 (enforcing)
```

You can also use `virsh` to display the interfaces that the vMX has in use by using the `domiflist` command.

For VCP:

```
user@host:~/vmx# sudo virsh domiflist vcp-vmx1
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vcp-ext-vmx1 bridge     br-ext     virtio      0a:00:dd:c0:de:0e
vcp-int-vmx1 bridge     br-int-vmx1 virtio      52:54:00:ca:43:a9
```

For VFP:

```
user@host:~/vmx# sudo virsh domiflist vfp-vmx1

Interface  Type       Source     Model       MAC
-------------------------------------------------------
vfp-ext-vmx1 bridge     br-ext     virtio      0a:00:dd:c0:de:10
vfp-int-vmx1 bridge     br-int-vmx1 virtio      52:54:00:0f:8f:37
ge-0.0.0-vmx1 network   default    virtio      02:06:0a:0e:ff:f0
```

Here you can see the Linux bridges and interfaces on each VM. As expected, the VCP and VFP have the internal and external bridges set up, and the VFP shows the ge-0/0/0 data interface you created earlier.

Try It Yourself *Using virsh, see what else you can learn about the vMX virtual machines and interface configuration. Run virsh with the help option to see what other parameters and configuration can be displayed.*

Summary

You should now have all the information on how to build a vMX, manage it, and connect it to an instance, for lab purposes or otherwise.

Spend some time checking that everything is up and running because you are about to build a simple topology by adding a second vMX instance, which will be connected to the one that you just built!

It's called networking!

Chapter 3

Build a Simple Topology

In this chapter you will extend the single instance topology and create two vMXs. Later, as a lab exercise, you will create logical systems, and then, just to demonstrate the capability of the vMXs, you will go on to configure EVPN on that topology.

First, let's examine our basic topology – we are going to set up two vMX instances and initiate connectivity between them.

Lab Topology

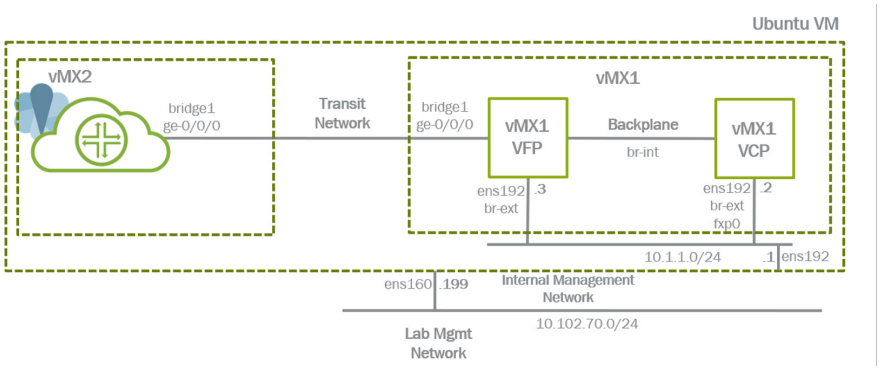

Figure 3.1 *Lab Topology*

As you can see in the topology, the management IP addresses have been set as 10.1.1.2 and 10.1.1.3 for the VCP and VFPs of vMX1. The ens192 interface is used for connecting the management interface. Similarly, let's set up the vMX2 instance.

Set Up a Second Instance of vMX

Remember there is a configuration file for a vMX instance in Chapter 2. Running a second vMX instance on a host is no different, and the second instance of vMX has its own settings file.

Copy vMX1's configuration file and use that as the basis for vMX2. If you've not already done so, it's time to SSH into the KVM host and go to the directory location where you installed the vMX:

```
user@host:~/vmx# cd config
user@host:~/vmx/config# cp vmx.conf vmx2.conf
```

Let's have a look at the settings to be changed in vmx2.conf.

When running multiple instances of vMX on the same host, each vMX instance needs to be configured with a unique identifier. Modify the configuration in vmx2.conf so that the vMX identifier is changed to vmx2.

This lab topology makes use of the same host management interface for both vMX1 and vMX2, and no changes need to be made to the images:

```
user@host:~/vmx/config# cat vmx2.conf
```

Let's examine the vMX2 configurations:

```
---
#Configuration on the host side - management interface, VM images etc.
HOST:
    identifier                : vmx2   # Maximum 6 characters
    host-management-interface : ens192
    routing-engine-image      : "/root/vmx/images/junos-vmx-x86-64-20.2R2.11.qcow2 "
    routing-engine-hdd        : "/root/vmx/images/vmxhdd.img"
    forwarding-engine-image   : "/root/vmx/images/vFPC-20201014.img"

---
```

The external bridge can be used by both vMX1 and vMX2 so no need to change this setting. Remember that this is used to bridge the management interfaces on the vMX to the host management interface defined above:

```
#External bridge configuration
BRIDGES:
    - type  : external
      name  : br-ext                 # Max 10 characters
```

For the VCP and VFP you will need to make some changes to the console port, the management IP address, and the MAC address.

The default MAC addresses used in the configuration file are taken from the locally administered MAC address ranges, so it is no problem to choose your own address from this range but take care not to overlap with the vMX1.

Don't forget to set a console port number and management IP address that will not overlap with vMX1. Here the management IP address is set to 10.1.1.4 and 10.1.1.5 for the VCP and VFPs:

```
#vRE VM parameters
CONTROL_PLANE:
    vcpus       : 1
    memory-mb   : 2048
    console_port: 8601

    interfaces  :
      - type      : static
        ipaddr    : 10.1.1.4
        macaddr   : "0A:00:DD:C0:DE:0E"
---
#vPFE VM parameters
FORWARDING_PLANE:
    memory-mb   : 4096
    vcpus       : 4
    console_port: 8602
    device-type : virtio

    interfaces  :
      - type      : static
        ipaddr    : 10.1.1.5
        macaddr   : "0A:00:DD:C0:DE:10"
#Interfaces
JUNOS_DEVICES:
   - interface            : ge-0/0/0
     mac-address          : "02:06:0A:0E:FF:F0"
     description          : "ge-0/0/0 interface"

#    - interface            : ge-0/0/1
#      mac-address          : "02:06:0A:0E:FF:F1"
#      description          : "ge-0/0/0 interface"

#    - interface            : ge-0/0/2
#      mac-address          : "02:06:0A:0E:FF:F2"
#      description          : "ge-0/0/0 interface"

#    - interface            : ge-0/0/3
#      mac-address          : "02:06:0A:0E:FF:F3"
#      description          : "ge-0/0/0 interface"
```

CAUTION We allocated 4 GB to the forwarding plane. This 4 GB should be fine for your lab purposes, depending on the features and version of vMX that you are using. Note that 1 GB should be the absolute minimum on the control plane. *Please don't do this in a production environment because it is not a Juniper supported configuration and if something goes wrong, JTAC won't help you!*

You should now uncomment ge-0/0/0 through ge-0/0/3 and again update the MAC addresses to ensure there's no clash with the vMX1:

```
---
interfaces :

     - link_name  : vmx_link1
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx1
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : bridge_dev
           dev_name    : bridge1

     - link_name  : vmx_link2
       mtu        : 1500
       endpoint_1 :
         - type        : junos_dev
           vm_name     : vmx2
           dev_name    : ge-0/0/0
       endpoint_2 :
         - type        : bridge_dev
           dev_name    : bridge1

    # - link_name  : vmx_link3
    #   endpoint_1 :
    #     - type        : junos_dev
    #       vm_name     : vmx1
    #       dev_name    : ge-0/0/1
    #   endpoint_2 :
    #     - type        : host_dev
    #       dev_name    : eth3

    # - link_name  : vmx_link4
    #   endpoint_1 :
    #     - type        : junos_dev
    #       vm_name     : vmx1
    #       dev_name    : ge-0/0/2
    #   endpoint_2 :
    #     - type        : junos_dev
    #       vm_name     : vmx2
    #       dev_name    : ge-0/0/2
```

Edit the configuration file (config/vmx.conf) and uncomment all four ge interfaces, then save the file:

```
#Interfaces
JUNOS_DEVICES:
   - interface            : ge-0/0/0
     mac-address          : "02:06:0A:0E:FF:F4"
     description          : "ge-0/0/0 interface"

   - interface            : ge-0/0/1
     mac-address          : "02:06:0A:0E:FF:F1"
     description          : "ge-0/0/1 interface"

   - interface            : ge-0/0/2
     mac-address          : "02:06:0A:0E:FF:F2"
     description          : "ge-0/0/2 interface"
```

```
  - interface           : ge-0/0/3
    mac-address         : "02:06:0A:0E:FF:F3"
    description         : "ge-0/0/3 interface"
```

Once you have saved the configuration file, the vMX2 is ready to be built. The same orchestration script that you used to create vMX1 is again used for vMX2, but this time you will need to specify an additional option to point the script at vMX2's configuration file.

CAUTION Each time you use vmx.sh to perform stop or start operations on vMX2, you must specify the configuration file for vMX2. Take care not to accidentally perform a stop operation on the wrong vMX! *In a production environment, you should not use the default configuration file locations.* This ensures that you must always specify a non-default configuration every time you execute the vmx.sh script.

Now enter the following command. The script will create the new vMX instance and will automatically start it for you:

```
user@host:~/vmx# sudo ./vmx.sh --install --cfg config/vmx2.conf

================================================
    Welcome to VMX
================================================
Date.....................................08/20/20 21:46:53
VMX Identifier...........................vmx2
Config file................................/root/vmx/config/vmx2.conf
Build Directory............................/root/vmx/build/vmx2
Assuming kvm hypervisor.....................
Virtualization type.....................kvm
Junos Device type.......................virtio
Environment file............................/root/vmx/env/ubuntu_virtio.env
Junos Device Type.......................virtio
Initialize scripts......................[OK]
[OK]
[OK]
================================================
    VMX Environment Setup Completed
================================================
================================================
    VMX Install & Start
================================================
Linux distribution......................ubuntu
Check GRUB..............................[Disabled]
Installation status of qemu-kvm.........[OK]
Installation status of libvirt-bin......[OK]
Installation status of bridge-utils.....[OK]
Installation status of python...........[OK]
Installation status of libyaml-dev......[OK]
Installation status of python-yaml......[OK]
Installation status of numactl..........[OK]
Installation status of libnuma-dev......[OK]
Installation status of libparted0-dev...[OK]
```

```
Installation status of libpciaccess-dev...........[OK]
Installation status of libyajl-dev................[OK]
Installation status of libxml2-dev................[OK]
Installation status of libglib2.0-dev.............[OK]
Installation status of libnl-dev..................[OK]
Check Kernel Version..............................[Disabled]
Check Qemu Version................................[Disabled]
Check libvirt Version.............................[Disabled]
Check virsh connectivity..........................[OK]
[OK]
[OK]
================================================
    Pre-Install Checks Completed
================================================
Check RE state....................................[Not Running]
[OK]
Check for VM vfp-vmx2.............................[Not Running]
[OK]
Check if bridge br-ext exists.....................[Yes]
Get Configured Management Interface...............ens192
Find existing management gateway..................ens160
Mgmt interface needs reconfiguration..............[No]
Cleanup VM bridge br-ext..........................[OK]
Cleanup VM bridge br-int-vmx2.....................[OK]
Cleanup VM bridge br-fab-vmx2.....................[OK]
================================================
    VMX Stop Completed
================================================
Check VCP image...................................[OK]
Check VFP image...................................[OK]
Check VCP Config image............................[OK]
Check management interface........................[OK]
Setup huge pages to 8192..........................[Already configured]
[OK]
Attempt to kill libvirtd..........................[OK]
Attempt to start libvirt-bin......................[OK]
Sleep 2 secs......................................[OK]
Check libvirt support for hugepages...............[OK]
================================================
    System Setup Completed
================================================
Generate libvirt files............................[OK]
Sleep 2 secs......................................[OK]
Find configured management interface..............ens192
Find existing management gateway..................ens160
Check if ens192 is already enslaved to br-ext.....[Yes]
Create br-int-vmx2................................[OK]
Start br-int-vmx2.................................[OK]
[OK]
Define vcp-vmx2...................................[OK]
Start vcp-vmx2....................................[OK]
Define vfp-vmx2...................................[OK]
Wait 2 secs.......................................[OK]
Start vfp-vmx2....................................[OK]
Wait 2 secs.......................................[OK]
================================================
    VMX Bringup Completed
================================================
```

```
Check if br-ext is created.....................[Created]
Check if br-int-vmx2 is created................[Created]
Check if VM vcp-vmx2 is running................[Running]
Check if VM vfp-vmx2 is running................[Running]
Check if tap interface vfp-ext-vmx2 exists.....[OK]
Check if tap interface vfp-int-vmx2 exists.....[OK]
Check if tap interface vcp-ext-vmx2 exists.....[OK]
Check if tap interface vcp-int-vmx2 exists.....[OK]
================================================
    VMX Status Verification Completed.
================================================
================================================
    Thank you for using VMX
================================================
```

Check the configured Linux bridges again:

```
user@host:~/vmx# brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.00505693ce43       yes             br-ext-nic
                                                        ens192
                                                        vcp-ext-vmx1
                                                        vcp-ext-vmx2
                                                        vfp-ext-vmx1
                                                        vfp-ext-vmx2
br-int-vmx1     8000.5254006895df       yes             br-int-vmx1-nic
                                                        vcp-int-vmx1
                                                        vfp-int-vmx1
br-int-vmx2     8000.525400d972ba       yes             br-int-vmx2-nic
                                                        vcp-int-vmx2
                                                        vfp-int-vmx2
bridge1         8000.fe060a0efff0       no              ge-0.0.0-vmx1
                                                        ge-0.0.0-vmx2
virbr0          8000.5254001bd668       yes             virbr0-nic
```

You can see that the vMX script automatically created another internal bridge named `br-int-vmx2`. This time the internal bridge is present to enable the VCP and VFP communication for vMX2. The external bridge (management bridge) is shared by all vMX management interfaces.

There are a couple of error messages that you might see if things didn't go well during the deployment of vMX. For instance, the next example shows that the console ports assigned to vMX1 and vMX2 are the same:

```
Start vcp-vmx2 [Failed]
error: Failed to start domain vcp-vmx2
error: internal error: process exited while connecting to monitor: 2020-08-16T21:09:18.408436Z qemu-
system-x86_64: -chardev socket,id=charserial0,host=127.0.0.1,port=8601,telnet,server,
no wait: Failed to bind socket: Address already in use
```

This next error message shows that there isn't enough system memory to start the VCP virtual machine:

```
Start vfp-vmx2 [Failed]
error: Failed to start domain vfp-vmx2
error: internal error: early end of file from monitor: possible problem: CPU feature invtsc not found
CPU feature invtsc not found CPU feature invtsc not found
file_ram_alloc: can't mmap RAM pages: Cannot allocate memory
```

If you remember when vMX1 was deployed in Chapter 2, only one ge- interface was configured. Before going any further in this lab, you will need to add the additional interfaces to vMX1. But first, use the libvirt `virsh` CLI command to compare the vMX1 with vMX2.

Use the `list` command to show the VMs (domains) that are configured. You can then use the `domiflist` command to show all the configured interfaces. If you are interested in the forwarding plane interfaces, you will need to query the correct domain ID, here 16 and 18:

```
user@host:~/vmx/config# sudo virsh list
 Id    Name                           State
----------------------------------------------------
 15    vcp-vmx2                       running
 16    vfp-vmx2                       running
 17    vcp-vmx1                       running
 18    vfp-vmx1                       running

user@host:~/vmx/config# sudo virsh domiflist 16
Interface  Type       Source     Model       MAC
-------------------------------------------------------------
vfp-ext-vmx2 bridge     br-ext     virtio      0a:00:dd:c0:de:10
vfp-int-vmx2 bridge     br-int-vmx2 virtio      52:54:00:36:f4:7c
ge-0.0.0-vmx2 network    default    virtio      02:06:0a:0e:ff:f0
ge-0.0.1-vmx2 network    default    virtio      02:06:0a:0e:ff:f1
ge-0.0.2-vmx2 network    default    virtio      02:06:0a:0e:ff:f2
ge-0.0.3-vmx2 network    default    virtio      02:06:0a:0e:ff:f3

user@host:~/vmx/config# sudo virsh domiflist 18
Interface  Type       Source     Model       MAC
-------------------------------------------------------------
vfp-ext-vmx1 bridge     br-ext     virtio      0a:00:dd:c0:de:10
vfp-int-vmx1 bridge     br-int-vmx1 virtio      52:54:00:ab:3e:1b
ge-0.0.0-vmx1 network    default    virtio      02:06:0a:0e:ff:f0
ge-0.0.1-vmx1 network    default    virtio      02:06:0a:0e:ff:f1
ge-0.0.2-vmx1 network    default    virtio      02:06:0a:0e:ff:f2
ge-0.0.3-vmx1 network    default    virtio      02:06:0a:0e:ff:f3
```

NOTE Whenever you make changes to the configuration, just saving the file will not make any changes to a running instance of vMX. You need to stop the running instance of vMX1, and then redeploy the instance:

1. Connect to the console on vMX1's VCP and stop Junos. Use the `request system halt` command to gracefully shut down the Junos software.
2. Stop the running instance (`sudo ./vmx.sh --stop`)
3. Re-deploy vMX1 (`sudo ./vmx.sh --install`)

The vMX will now be restarted with the additional interfaces.

Okay, you're now ready to connect to the console on the vMX2. This is done the same way for vMX1 and vMX2. You simply reference the correct vMX instance when running the script. If you wish, now would be a good time to configure SSH access to vMX2:

```
user@host:~/vmx# sudo ./vmx.sh --console vcp vmx2
--
Login Console Port For vcp-vmx2 - 8603
Press Ctrl-] to exit anytime
--
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

FreeBSD/amd64 (Amnesiac) (ttyu0)

login: root

--- JUNOS 20.2R2.11 Kernel 64-bit  JNPR-11.0-20200608.0016468_buil
root@:~ #
```

Next configure the vMX2 Instance to set an IP address on the management interface:

```
set interfaces fxp0 unit 0 family inet address 10.1.1.4/24
```

Enable the SSH service:

```
set system services ssh
set system services ssh allow-root-login
```

Set the hostname and a password for the root user, if you have not done so already:

```
set system host-name vmx2
set system root-authentication plain-text-password
```

Commit the configuration and exit the console session. You should now be able to SSH directly to the vMX using the IP address that was just configured on the fxp0 interface:

```
user@host:~/vmx# ssh root@10.1.1.4

The authenticity of host 10.1.1.4 (10.1.1.4) can't be established.
ECDSA key fingerprint is SHA256:dqlbxXuQbqSJd9oLQynvig3IbvbNWxW4laAVix5SryY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.1.1.4' (ECDSA) to the list of known hosts.
Password:
Last login: Fri Aug 21 05:10:03 2020
--- JUNOS 20.2R2.11 Kernel 64-bit  JNPR-11.0-20200608.0016468_buil
root@vmx2:~ #
```

It's now time to connect vMX1 and vMX2 and build the sample topology.

Link Two vMXs with Virtio

For the Ethernet connectivity to the vMX you will be using KVM virtio paravirtualization. Virtio bindings are flexible and can be used to map multiple vMX instances to a physical host interface, or to connect vMX instances or vMX interfaces together, which you'll be doing here. Linux bridges are used to stitch everything together.

A Linux bridge is a way to connect two or more Ethernet segments in software. Think of it as a virtual network switch. Packets are forwarded based on the MAC address, and both untagged and 802.1q tagged frames are supported.

At this point both the vMX1 and vMX2 are running but you need to create the virtio bindings to enable the communication between each vMX.

For both vMX1 and vMX2 this is done in the same configuration file – config/vmx-junosdev.conf. The goal is to connect interfaces ge-0/0/1 to ge-0/0/2 back-to-back on each vMX and link together the vMX instances using ge-0/0/3 on each vMX.

Create a link between vMX1 interfaces ge-0/0/1 and ge-0/0/2:

```
  - link_name  : vmx_link3
  mtu        : 1500
  endpoint_1 :
    - type       : junos_dev
      vm_name    : vmx1
      dev_name   : ge-0/0/1
  endpoint_2 :
    - type       : junos_dev
      vm_name    : vmx1
      dev_name   : ge-0/0/2
```

The same is done for vMX2:

```
- link_name  : vmx_link4
  mtu        : 1500
  endpoint_1 :
    - type       : junos_dev
      vm_name    : vmx2
      dev_name   : ge-0/0/1
  endpoint_2 :
    - type       : junos_dev
      vm_name    : vmx2
      dev_name   : ge-0/0/2
```

Finally, create a link between ge-0/0/3 on vMX1 and vMX2. You can use the same technique as shown above, but what if you want to connect more than two vMXs on the same Ethernet segment? That is done like this, with an additional bridge being defined and shared by each vMX:

```
- link_name  : bridge_vmx_l2
  mtu        : 1500
```

```
      endpoint_1 :
        - type       : junos_dev
          vm_name    : vmx1
          dev_name   : ge-0/0/3
      endpoint_2 :
        - type       : bridge_dev
          vm_name    : vmx1
          dev_name   : bridge_vmx12
    - link_name  : bridge_vmx_12
      mtu        : 1500
      endpoint_1 :
        - type       : junos_dev
          vm_name    : vmx2
          dev_name   : ge-0/0/3
      endpoint_2 :
        - type       : bridge_dev
          vm_name    : vmx2
          dev_name   : bridge_vmx12
```

Again the orchestration script vmx.sh is used to create the device bindings:

```
user@host:~/vmx# sudo ./vmx.sh --bind-dev
Checking package ethtool.........................[OK]
Bind Bridge port bridge1(ge-0.0.0-vmx1)...........[OK]
Bind Bridge port bridge1(ge-0.0.0-vmx2)...........[OK]
Bind Link vmx_link3(ge-0.0.1-vmx1, ge-0.0.2-vmx1)
[OK]
Bind Link vmx_link4(ge-0.0.1-vmx2, ge-0.0.2-vmx2)
[OK]
Bind Bridge port bridge_vmx12(ge-0.0.3-vmx1)......[OK]
Bind Bridge port bridge_vmx12(ge-0.0.3-vmx2)......[OK]
```

Now let's look at what bridges were created:

```
user@host:~/vmx# brctl show
bridge name     bridge id               STP enabled     interfaces
br-ext          8000.00505693ce43       yes             br-ext-nic
                                                        ens192
                                                        vcp-ext-vmx1
                                                        vcp-ext-vmx2
                                                        vfp-ext-vmx1
                                                        vfp-ext-vmx2
br-int-vmx1     8000.525400e67ed3       yes             br-int-vmx1-nic
                                                        vcp-int-vmx1
                                                        vfp-int-vmx1
br-int-vmx2     8000.52540088f32e       yes             br-int-vmx2-nic
                                                        vcp-int-vmx2
                                                        vfp-int-vmx2
bridge1         8000.fe060a0efff0       no              ge-0.0.0-vmx1
                                                        ge-0.0.0-vmx2
bridge_vmx12    8000.fe060a0efff3       no              ge-0.0.3-vmx1
                                                        ge-0.0.3-vmx2
bridge_vmx_l2   8000.000000000000       no
virbr0          8000.5254001bd668       yes             virbr0-nic
vmx_link3       8000.fe060a0efff1       no              ge-0.0.1-vmx1
                                                                ge-0.0.2-vmx1
vmx_link4       8000.fe060a0efff1       no              ge-0.0.1-vmx2
                                                        ge-0.0.2-vmx2
```

Table 3.1 contains descriptions for each bridge.

Table 3.1 Description of Bridges

Components	Description
br-ext	The external bridge for management traffic
br-int-vmx1	The internal bridge for vMX1 RE to PFE traffic
br-int-vmx2	The internal bridge for vMX2 RE to PFE traffic
bridge1	Enables the communication between ge-0/0/0 on vMX1 and vMX2
bridge _ vmx12	Enables the communication between ge-0/0/3 on vMX1 and vMX2
virbr0	This default KVM bridge is unused as all vMX interfaces are defined (not shown above)
vmx_link3	Connects ge-0/0/1 and ge-0/0/2 on vMX1
vmx_link4	Connects ge-0/0/1 and ge-0/0/2 on vMX2

At this point vMX1 and vMX2 are ready to be configured. What better way to test your two vMXs than a quick lab build!

EVPN Lab

In this lab you will create the following simple topology of four MX routers. You will be able to extend the principles shown here to expand your own topology to be as large and complex as you like. A more detailed topology will be used in Chapter 4.

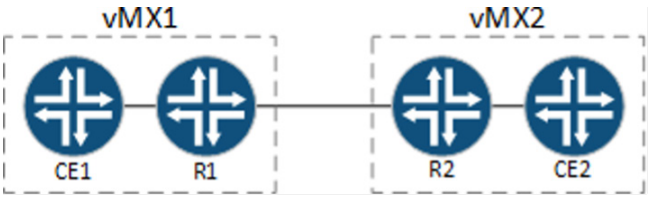

Figure 3.2 Lab Topology

Figure 3.2's topology consists of two vMXs running on the same Ubuntu host. You will create CE1 and CE2 as logical system routers. In the topology you will also configure EVPN, however EVPN is unfortunately not supported within a logical system, so R1 and R2 will be the main routers on each vMX and will also be your EVPN PEs.

EVPN is defined in RFC7432. It provides a number of enhancements over VPLS, particularly as MAC address learning now occurs in the control plane and is advertised between PEs using an MP-BGP MAC route. Compared to VPLS, which uses data plane flooding to learn MAC addresses, this BGP-based approach enables EVPN to limit the flooding of unknown unicast. MAC addresses are now being routed, which in multihomed scenarios enables all active links to be utilized. Neat stuff.

MORE? See *Day One: Using Ethernet VPNs for Data Center Interconnect* at http://www.juniper.net/us/ en/training/jnbooks/day-one/proof-concept-labs/using-ethernet-vpns/.

You will now create a topology that makes use of the virtio bindings that were created earlier in this chapter. To recap, ge-0/0/1 and ge-0/0/2 are connected back-to-back on vMX1 and vMX2. Then vMX1 and vMX2 are connected via ge-0/0/3. In terms of this topology:

- R1 ge-0/0/3 connects to R2 ge-0/0/3
- CE1 ge-0/0/2 (VLAN 34) connects to R1 ge-0/0/1 (VLAN 34)
- CE2 ge-0/0/2 (VLAN 34) connects to R2 ge-0/0/1 (VLAN 34)

R1 and R2 represent your core routers and as such will be running MPLS. You will configure EVPN on R1 and R2 and use EVPN to create a Layer 2 connection between CE1 and CE2.

You can consider this lab a success if CE1 and CE2 view each other as directly adjacent and if you are able to ping between CE1 and CE2.

NOTE This *Day One* book is about building up your lab topology using vMX, so detail on EVPN will be at a high level. If you would like to know more about EVPN then check out *Day One: Using Ethernet VPNs for Data Center Interconnect* at http://www.juniper.net/us/ en/training/jnbooks/day-one/proof-concept-labs/using-ethernet-vpns/.

Lab Configuration

If you have not already applied a trial license to vMX2 you should refer back to Chapter 2 and apply a trial license now, before continuing any further.

First, apply a base configuration to R1 and R2 and then test the connectivity. In the base configuration, R1 and R2 should use OSPF as the IGP. Also, you will need MPLS so enable family MPLS and LDP on interface ge-0/0/3. For R1, use a loopback IP of 1.1.1.1/32 and ge-0/0/3.0 as 192.168.12.1/30.

For R2 use a loopback IP of 2.2.2.2/32 and ge-0/0/3.0 as 192.168.12.2/30.

On vMX1:

```
set system host-name R1
set interfaces ge-0/0/3 unit 0 family inet address 192.168.12.1/30
set interfaces ge-0/0/3 unit 0 family mpls
set interfaces lo0 unit 0 family inet address 1.1.1.1/32
set routing-options router-id 1.1.1.1
set protocols mpls interface ge-0/0/3.0
set protocols ospf area 0.0.0.0 interface lo0.0 passive
set protocols ospf area 0.0.0.0 interface ge-0/0/3.0
set protocols ldp interface ge-0/0/3.0
```

On vMX2:

```
set system host-name R2
set interfaces ge-0/0/3 unit 0 family inet address 192.168.12.2/30
set interfaces ge-0/0/3 unit 0 family mpls
set interfaces lo0 unit 0 family inet address 2.2.2.2/32
set routing-options router-id 2.2.2.2
set protocols mpls interface ge-0/0/3.0
set protocols ospf area 0.0.0.0 interface lo0.0 passive
set protocols ospf area 0.0.0.0 interface ge-0/0/3.0
set protocols ldp interface ge-0/0/3.0
```

Don't forget to set a password for the root account before you commit the configuration:

```
set system root-authentication plain-text-password
```

Next check the status of the OSPF neighbors. If everything is up you should be able to ping between the two loopback addresses:

```
user@host> show ospf neighbor
Address    Interface  State  IDPri Dead
192.168.12.2 ge-0/0/3.0 Full2.2.2.2  128  39

user@host> ping 2.2.2.2 rapid source 1.1.1.1
PING 2.2.2.2 (2.2.2.2): 56 data bytes
!!!!!
--- 2.2.2.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss round-trip min/avg/max/
stddev = 1.018/1.299/1.685/0.223 ms
```

Now that you have reachability between R1 and R2 you can go ahead and add the required base configuration for EVPN.

NOTE Unfortunately, EVPN is not supported within a logical system, which is why you will need to configure EVPN on the main routers.

Now configure MP-BGP making sure to activate the `evpn signaling` MP-BGP address family. As this EVPN configuration is Layer 2 only, the `inet-vpn unicast` MP-BGP address family is optional. To configure the iBGP peering between R1 and R2, use AS65000 as your Autonomous System.

Configure the BGP peering between each loopback address on R1 (vMX1):

```
set routing-options autonomous-system 65000
set protocols bgp group internal type internal
set protocols bgp group internal local-address 1.1.1.1
set protocols bgp group internal family inet-vpn unicast
set protocols bgp group internal family evpn signaling
set protocols bgp group internal neighbor 2.2.2.2
```

On R2 (vMX2):

```
set routing-options autonomous-system 65000
set protocols bgp group internal type internal
set protocols bgp group internal local-address 2.2.2.2
set protocols bgp group internal family inet-vpn unicast
set protocols bgp group internal family evpn signaling
set protocols bgp group internal neighbor 1.1.1.1
```

Make sure that the neighborship is established, but of course you will not see any routes received or advertised at this point:

```
user@host> show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table Tot Paths Act Paths Suppressed History Damp State Pending
bgp.l3vpn.0
             0  0 0  0 0 0
bgp.evpn.0
             0 0 0 0 0 0
Peer AS InPkt OutPkt OutQ Flaps Last Up/
Dwn State|#Active/Received/Accepted/Damped...
2.2.2.2     65000   4  4  0  0  32 Establ
bgp.l3vpn.0: 0/0/0/0
bgp.evpn.0: 0/0/0/0
```

Logical Systems

The configuration gets a little more complicated here because you need to create CE1 and CE2 as logical system routers on each vMX. Remember that ge-0/0/1 and ge0/0/2 have been connected back-to-back by the virtio bridge. Use ge-0/0/1 as the interface on R1/R2, and ge-0/0/2 as the interfaces on the logical system routers CE1/CE2.

Configure your topology as follows:

1. Create a logical system named CE1 on R1, assigning interface ge-0/0/2. Configure the IP 192.168.34.3/29 on ge-0/0/2. Use a VLAN ID of 34.
2. Create a logical system named CE2 on R2, assigning interface ge-0/0/2. Configure the IP 192.168.34.4/29 on ge-0/0/2. Use a VLAN ID of 34.

In Chapter 4 you will see much more on logical system routers.

On R1 (vMX1):

```
set logical-systems CE1 interfaces ge-0/0/2 unit 34 vlan-id 34
set logical-systems CE1 interfaces ge-0/0/2 unit 34 family inet address 192.168.34.3/29
set interfaces ge-0/0/2 vlan-tagging
```

On R2 (vMX2):

```
set logical-systems CE2 interfaces ge-0/0/2 unit 34 vlan-id 34
set logical-systems CE2 interfaces ge-0/0/2 unit 34 family inet address 192.168.34.4/29
set interfaces ge-0/0/2 vlan-tagging
```

Working with these logical systems is simple and commands can be entered in a couple of ways. Configuration can also be entered directly when the CLI is set to a logical system. Here are two ways to ping CE1's own interface:

```
user@host> set cli logical-system CE1
logical system: CE1

user@host:CE1> ping 192.168.34.3 rapid

PING 192.168.34.3 (192.168.34.3): 56 data bytes
!!!!!
--- 192.168.34.3 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss round-trip min/avg/max/
stddev = 0.005/0.010/0.020/0.006 ms

user@host:CE1> clear cli logical-system

Cleared default logical system

user@host> ping logical-system CE1 192.168.34.3 rapid

PING 192.168.34.3 (192.168.34.3): 56 data bytes
!!!!!
--- 192.168.34.3 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss round-trip min/avg/max/
stddev = 0.004/0.008/0.017/0.005 ms
```

Clearly at this point CE1 and CE2 will not be able to ping each other because you need to use EVPN to provide the Layer 2 connectivity.

Completing the EVPN Configuration

Now let's configure the EVPN VLAN-based service. This requires a separate EVI per VLAN. An EVI is an EVPN instance spanning across the PEs participating in a particular EVPN. There isn't too much to the configuration.

On R1 and R2 configure the interface-facing CE1 (ge-0/0/1) to support `vlan-tagging` and with `flexible-ethernet-services` encapsulation. Then configure unit 34 with the correct `vlan-id` and `vlan-bridge` encapsulation. You will also need to define the EVPN routing instance itself. Interface ge-0/0/1.34 (the interface facing the CE router) is added to the EVPN instance. Here is the sample configuration for R1 and R2:

```
set interfaces ge-0/0/1 flexible-vlan-tagging
set interfaces ge-0/0/1 encapsulation flexible-ethernet-services
set interfaces ge-0/0/1 unit 34 encapsulation vlan-bridge
set interfaces ge-0/0/1 unit 34 vlan-id 34
set routing-instances EVPN34 instance-type evpn
set routing-instances EVPN34 vlan-id 34
set routing-instances EVPN34 interface ge-0/0/1.34
set routing-instances EVPN34 route-distinguisher 1.1.1.1:1
set routing-instances EVPN34 vrf-target target:34:34
set routing-instances EVPN34 protocols evpn
```

Verification

At this point the configuration of EVPN is complete so let's verify that everything is working as expected. On the EVPN PE routers, check that the routes are being received in BGP:

```
user@host> show bgp summary
Groups: 1 Peers: 1 Down peers: 0
Table      Tot Paths Act Paths Suppressed  History Damp State  Pending bgp.l3vpn.0

bgp.evpn.0

0          0 0 0 0 0

1          1 0 0 0 0

Peer       ASInPkt  OutPkt OutQ Flaps Last Up/Dwn State|#Active/ Received/Accepted/Damped...
2.2.2.2    65000  83550 0 20:34 Establ
bgp.l3vpn.0: 0/0/0/0 bgp.evpn.0: 1/1/1/0 EVPN34.evpn.0: 1/1/1/0
  default_evpn .evpn.0: 0/0/0/0
```

This looks good – one route received. As previously mentioned, this configuration is Layer 2 only, so table bgp. l3vpn.0 remains empty.

Can CE1 and CE2 now ping each other? Let's check:

```
user@host> set cli logical-system CE1 Logical system: CE1

user@host:CE1> ping 192.168.34.4 rapid
PING 192.168.34.4 (192.168.34.4): 56 data bytes
!!!!!
--- 192.168.34.4 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.432/49.696/200.211/76.672 ms

user@host:CE1> show arp
MAC Address    Address        Name       Interface Flags
02:06:0a:0e:ff:f6  192.168.34.4 192.168.34.4 ge-0/0/2.34  none
```

Looks good! Notice that the CE2 MAC address is in CE1's ARP table.

Now for a little more detail on what the EVPN PEs are seeing. Connect back to R1. You should be able to see the MAC addresses in the BGP table, the directly-attached device, but also the device attached to R3:

```
user@host> show route table EVPN34.evpn.0

EVPN34.evpn.0: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

2:1.1.1.1:1::34::02:06:0a:0e:ff:f2/304
*[EVPN/170] 00:01:52
Indirect 2:1.1.1.1:1::34::02:06:0a:0e:ff:f6/304
*[BGP/170] 00:01:52, localpref 100, from 2.2.2.2 AS path: I, validation-state: unverified
> to 192.168.12.2 via ge-0/0/3.0 3:1.1.1.1:1::34::1.1.1.1/304
*[EVPN/170] 00:04:04
Indirect 3:1.1.1.1:1::34::2.2.2.2/304
*[BGP/170] 00:02:54, localpref 100, from 2.2.2.2 AS path: I, validation-state: unverified
> to 192.168.12.2 via ge-0/0/3.0
```

If you would like to view the compete EVPN database and MAC table, use the `show evpn database` command:

```
user@host> show evpn database

Instance: EVPN34
VLAN MAC address Active source Timestamp IP address
34 02:06:0a:0e:ff:f2 ge-0/0/1.34 Feb 23 15:03:42
34 02:06:0a:0e:ff:f6 2.2.2.2 Feb 23 15:05:24


user@host> show evpn mac-table

MAC flags (S -static MAC, D -dynamic MAC, L -locally learned, C -Control MAC
O -OVSDB MAC, SE -Statistics enabled, NM -Non configured MAC, R -Remote PE MAC)
Routing instance : EVPN34
Bridging domain : __EVPN34__, VLAN : 34
MAC MAC Logical NH RTR
address flags interface Index ID
02:06:0a:0e:ff:f2 D ge-0/0/1.34
02:06:0a:0e:ff:f6 DC 1048575 1048575
```

You can also check that local MAC addresses are being advertised from R1 to R2:

```
user@host> show route advertising-protocol bgp 2.2.2.2

EVPN34.evpn.0: 4 destinations, 4 routes (4 active, 0 holddown, 0 hidden)
Prefix       Nexthop  MED Lclpref  AS path
2:1.1.1.1:1::34::02:06:0a:0e:ff:f2/304
* Self 100 I
3:1.1.1.1:1::34::1.1.1.1/304
* Self 100 I
```

Or view detailed information about the EVPN routing instance. There is some useful information here, for example, the total number of MAC addresses:

```
user@host> show evpn instance EVPN34 extensive Instance: EVPN34
Route Distinguisher: 1.1.1.1:1
VLAN ID: 34
Per-instance MAC route label: 299808
MAC database status    Local Remote
Total MAC addresses:      1 1
```

```
Default gateway MAC addresses:  0 0
Number of local interfaces: 1 (1 up)
Interface name ESI        Mode Status
ge-0/0/1.34   00:00:00:00:00:00:00:00:00:00 single-homed Up
Number of IRB interfaces: 0 (0 up) Number of bridge domains: 1
VLAN ID Intfs / up Mode MAC sync IM route label
34         1 1 Extended Enabled 299872
Number of neighbors: 1 2.2.2.2
Received routes
MAC address advertisement:  1
MAC+IP address advertisement: 0
Inclusive multicast: 1
Ethernet auto-discovery:  0
Number of ethernet segments: 0
```

Monitor the Traffic Going Between the Two Logical System Routers

Finally, as you have already used Linux bridges to interconnect the logical system routers, you can use tcpdump on the Linux host to monitor traffic between each logical system router. This is achieved by monitoring the Linux bridge:

```
sudo tcpdump -i vmx1_link_ls -n
```

Summary

In this chapter you have hopefully discovered how simple it is to deploy and interconnect multiple vMXs on the same Linux host. And you just built a topology of four logical routers on the two vMXs and used EVPN to demonstrate the extensive capability of vMX.

Time to rock. Let's scale it.

Chapter 4

Scaling Your vMX Topology

Now let's scale your lab topology. There are a couple of options here, depending on your own preference and the amount of capacity that you have to spare on your KVM host.

The obvious option for scaling a lab using vMX routers is simply to run more vMX instances on the same host. If you have the hardware available, then this is certainly a good choice. Using the principles you've learned throughout this book, you can simply add more vMX instances and connect interfaces together using virtio bindings and Linux bridges. This is a very flexible and simple way to build a large topology.

But what if your lab hardware specification is not enough to run several vMXs on the same host? Well, you can use fewer vMXs and make extensive use of virtual routers or logical systems as shown in Chapter 3. In this chapter you will scale out our topology of two vMXs using logical systems. With just two vMXs and use of logical systems, you could create a topology of thirty routers.

Just for fun, a Linux VM will be added to the topology and it will be configured as a CE, but also with a BGP route server installed.

Let's get started.

Lab Topology

You will need to create three VMs to complete this lab, two vMX instances and an Ubuntu Linux virtual machine as shown in Figure 4.1. And Table 4.1 lists how each VM interface will be configured.

Figure 4.1 *Lab Topology*

Table 4.1 *Lab Interface Configuration*

VM	Interface	Connects to	Note
CE1	ens192	vMX1 ge-0/0/0	
vMX1	ge-0/0/0	CE1 eth1	
vMX1	ge-0/0/1	vMX1 ge-0/0/2	Used for logical system communication
vMX1	ge-0/0/2	vMX1 ge-0/0/1	Used for logical system communication
vMX1	ge-0/0/3	vMX2 ge-0/0/3	Used for vMX instance communication
vMX2	ge-0/0/3	vMX1 ge-0/0/3	Used for vMX instance communication

Communication between logical systems can be done in a couple of different ways. In Chapter 3, communication between logical systems was accomplished using *Ethernet interfaces and VLANs*. Two vMX interfaces were connected back-to-back using virtio and Linux bridges to link the interfaces together. One end of the link was placed in one logical system and the other end was placed in a different logical system.

The same thing can also be accomplished using *logical tunnel interfaces,* which simply creates a set of logical point-to-point interfaces on the vMX. One end of the link is placed in one logical system, and the other end of the point-to-point is placed in a different logical system. As with a physical interface, you can run dynamic routing protocols over the tunnel if you wish. Logical tunnels are a really flexible way to leak routes between routing instances or logical systems. They've probably helped you out a few times!

In this lab, the vMX1 will use Ethernet interfaces to join the logical systems and the vMX2 will use logical tunnels to join the logical systems.

Logical Topology

Figure 4.2 illustrates the topology you will create. The network will run OSPF as the IGP – and MPLS of course – but this time it's RSVP signaled. The goal is to enable VPLS over this topology and use VPLS to link CE1 and CE2. You could do a simple Layer 2 circuit point-to-point link, but what if your customer tells you they expect to add additional sites later and these sites all need to be directly reachable at Layer 2? So VPLS it is. EVPN would also work, but you already configured EVPN in Chapter 3.

CE1 is the Linux host configured as a BGP route server, while CE2 will be a Junos logical system MX. You'll configure the route server and a BGP peering between CE1 and CE2. The logical topology is shown in Figure 4.2.

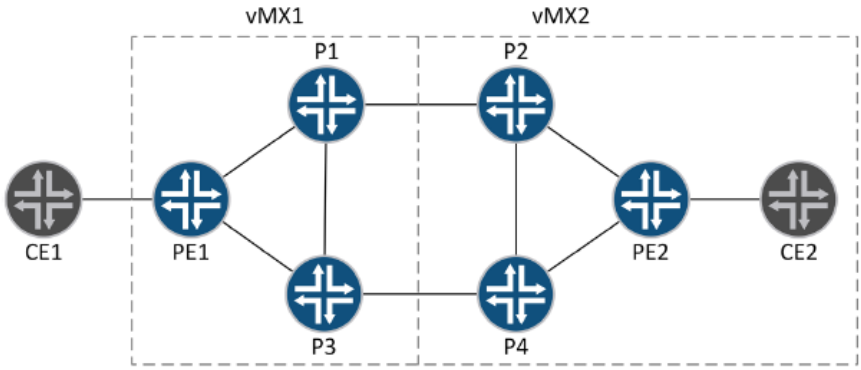

Figure 4.2 Logical Topology

The interface and IP address schema are listed in Table 4.2.

Table 4.2 Interface and IP Configuration of Figure 4.2

Router	Interface	VLAN ID / Peer unit	Connects to	IP address
P1	ge-0/0/3.12	12	P2	192.168.12.1/30
P1	ge-0/0/1.13	13	P3	192.168.13.1/30
P1	ge-0/0/1.15	15	PE1	192.168.15.1/30
P1	Lo0.1			10.1.1.1/32
P2	ge-0/0/3.12	12	P1	192.168.12.2/30
P2	lt-0/0/0.242	244	P4	192.168.24.1/30

P2	lt-0/0/0.262	266	PE2	192.168.26.1/30
P2	Lo0.2			10.2.2.2/32
P3	ge-0/0/2.13	13	P1	192.168.13.2/30
P3	ge-0/0/3.34	34	P4	192.168.34.1/30
P3	ge-0/0/1.35	35	PE1	192.168.35.1/30
P3	Lo0.3			10.3.3.3/32
P4	lt-0/0/0.244	242	P2	192.168.24.2/30
P4	ge-0/0/3.34	34	P3	192.168.34.2/30
P4	lt-0/0/0.464	466	PE2	192.168.46.1/30
P4	Lo0.4			10.4.4.4/32
PE1	ge-0/0/2.15	15	P1	192.168.15.2/30
PE1	ge-0/0/2.35	35	P3	192.168.35.2/30
PE1	ge-0/0/0		CE1	N/A
PE1	Lo0.5			10.5.5.5/32
PE2	lt-0/0/0.266	262	P2	192.168.26.2/30
PE2	lt-0/0/0.466	464	P4	192.168.46.2/30
PE2	lt-0/0/0.686	688	CE2	N/A
PE2	Lo0.6			10.6.6.6/32
CE1	ens192		PE1	10.0.0.1/24
CE2	lt-0/0/0.688	686	PE2	10.0.0.2/24

Lab vMX Configuration

You will reuse the vMX1 and vMX2 configuration files from the previous chapters.

Before getting started, you will need to start both vMX and reset the Junos configuration to factory defaults:

```
user@R1> configure
Entering configuration mode

[edit]
user@R1# load factory-default
warning: activating factory configuration
```

Base Configuration for the P/PE Core

Now that each vMX is at defaults, apply the logical system and IP address configurations as shown in Table 4.2. Let's configure the vMX1 first. On vMX1 the LS routers will be connected together using VLAN tagged interfaces.

1. Set the host name to vMX1 and configure a password for the root account:

```
set system host-name vmx1
set system root-authentication plain-text-password
```

2. Enable support for VLAN-tagging on ge-0/0/1 through ge-0/0/3:

```
set interfaces ge-0/0/1 flexible-vlan-tagging
set interfaces ge-0/0/2 flexible-vlan-tagging set interfaces ge-0/0/3 flexible-vlan-tagging
```

3. Create the logical systems. Assign the interfaces and IPs to each P/PE router and enable MPLS support on the interface:

```
set logical-systems P1 interfaces ge-0/0/1 unit 13 vlan-id 13
set logical-systems P1 interfaces ge-0/0/1 unit 13 family inet address 192.168.13.1/30
set logical-systems P1 interfaces ge-0/0/1 unit 13 family mpls
set logical-systems P1 interfaces ge-0/0/1 unit 15 vlan-id 15
set logical-systems P1 interfaces ge-0/0/1 unit 15 family inet address 192.168.15.1/30
set logical-systems P1 interfaces ge-0/0/1 unit 15 family mpls
set logical-systems P1 interfaces ge-0/0/3 unit 12 vlan-id 12
set logical-systems P1 interfaces ge-0/0/3 unit 12 family inet address 192.168.12.1/30
set logical-systems P1 interfaces ge-0/0/3 unit 12 family mpls
set logical-systems P1 interfaces lo0 unit 1 family inet address 10.1.1.1/32
set logical-systems P3 interfaces ge-0/0/1 unit 35 vlan-id 35
set logical-systems P3 interfaces ge-0/0/1 unit 35 family inet address 192.168.35.1/30

set logical-systems P3 interfaces ge-0/0/1 unit 35 family mpls
set logical-systems P3 interfaces ge-0/0/2 unit 13 vlan-id 13
set logical-systems P3 interfaces ge-0/0/2 unit 13 family inet address 192.168.13.2/30
set logical-systems P3 interfaces ge-0/0/2 unit 13 family mpls
set logical-systems P3 interfaces ge-0/0/3 unit 34 vlan-id 34
set logical-systems P3 interfaces ge-0/0/3 unit 34 family inet address 192.168.34.1/30
set logical-systems P3 interfaces ge-0/0/3 unit 34 family mpls
set logical-systems P3 interfaces lo0 unit 3 family inet address 10.3.3.3/32
set logical-systems PE1 interfaces ge-0/0/2 unit 15 vlan-id 15
set logical-systems PE1 interfaces ge-0/0/2 unit 15 family inet address 192.168.15.2/30
set logical-systems PE1 interfaces ge-0/0/2 unit 15 family mpls
set logical-systems PE1 interfaces ge-0/0/2 unit 35 vlan-id 35
set logical-systems PE1 interfaces ge-0/0/2 unit 35 family inet address 192.168.35.2/30
set logical-systems PE1 interfaces ge-0/0/2 unit 35 family mpls
set logical-systems PE1 interfaces lo0 unit 5 family inet address 10.5.5.5/32
```

Now configure vMX2. On vMX2 the LS routers will be connected using logical tunnel interfaces.

1. Set the host name to vMX2 and configure a password for the root account:

```
set system host-name vmx2
set system root-authentication plain-text-password
```

2. Enable support for VLAN-tagging on ge-0/0/1 through ge-0/0/3:

```
set interfaces ge-0/0/1 flexible-vlan-tagging
set interfaces ge-0/0/2 flexible-vlan-tagging
set interfaces ge-0/0/3 flexible-vlan-tagging
```

3. Configure FPC slot 0 to support logical tunnel (lt) interfaces. This creates a specific lt interface – make a note of this because you will need it when creating the lt units:

```
set chassis fpc 0 pic 0 tunnel-services

commit
commit complete

[edit]
user@vmx2# run show interfaces terse | match lt-
lt-0/0/0   upup
```

4. Create the logical systems. Assign the interfaces and IPs to each P/PE router and enable MPLS support on the interface. The configuration of the lt interface is simple. Create the lt interface and unit. The peer-unit specifies the lt peer-unit for the far end of the virtual point-to-point link:

```
set logical-systems P2 interfaces ge-0/0/3 unit 12 vlan-id 12
set logical-systems P2 interfaces ge-0/0/3 unit 12 family inet address 192.168.12.2/30
set logical-systems P2 interfaces ge-0/0/3 unit 12 family mpls
set logical-systems P2 interfaces lt-0/0/0 unit 242 encapsulation ethernet
set logical-systems P2 interfaces lt-0/0/0 unit 242 peer-unit 244
set logical-systems P2 interfaces lt-0/0/0 unit 242 family inet address 192.168.24.1/30
set logical-systems P2 interfaces lt-0/0/0 unit 242 family mpls
set logical-systems P2 interfaces lt-0/0/0 unit 262 encapsulation ethernet
set logical-systems P2 interfaces lt-0/0/0 unit 262 peer-unit 266
set logical-systems P2 interfaces lt-0/0/0 unit 262 family inet address 192.168.26.1/30
set logical-systems P2 interfaces lt-0/0/0 unit 262 family mpls
set logical-systems P2 interfaces lo0 unit 2 family inet address 10.2.2.2/32
set logical-systems P4 interfaces ge-0/0/3 unit 34 vlan-id 34
set logical-systems P4 interfaces ge-0/0/3 unit 34 family inet address 192.168.34.2/30
set logical-systems P4 interfaces ge-0/0/3 unit 34 family mpls
set logical-systems P4 interfaces lt-0/0/0 unit 244 encapsulation ethernet
set logical-systems P4 interfaces lt-0/0/0 unit 244 peer-unit 242
set logical-systems P4 interfaces lt-0/0/0 unit 244 family inet address 192.168.24.2/30
set logical-systems P4 interfaces lt-0/0/0 unit 244 family mpls
set logical-systems P4 interfaces lt-0/0/0 unit 464 encapsulation ethernet
set logical-systems P4 interfaces lt-0/0/0 unit 464 peer-unit 466
set logical-systems P4 interfaces lt-0/0/0 unit 464 family inet address 192.168.46.1/30
set logical-systems P4 interfaces lt-0/0/0 unit 464 family mpls
set logical-systems P4 interfaces lo0 unit 4 family inet address 10.4.4.4/32
set logical-systems PE2 interfaces lt-0/0/0 unit 266 encapsulation ethernet
set logical-systems PE2 interfaces lt-0/0/0 unit 266 peer-unit 262
set logical-systems PE2 interfaces lt-0/0/0 unit 266 family inet address 192.168.26.2/30
set logical-systems PE2 interfaces lt-0/0/0 unit 266 family mpls
set logical-systems PE2 interfaces lt-0/0/0 unit 466 encapsulation ethernet
set logical-systems PE2 interfaces lt-0/0/0 unit 466 peer-unit 464
set logical-systems PE2 interfaces lt-0/0/0 unit 466 family inet address 192.168.46.2/30
set logical-systems PE2 interfaces lt-0/0/0 unit 466 family mpls
set logical-systems PE2 interfaces lo0 unit 6 family inet address 10.6.6.6/32
```

For the routers that are interconnected with lt interfaces you will now be able to verify connectivity between each router with ping. But for routers on the vMX1, the Linux bridges and device bindings need to be set up before the logical system routers will be able to communicate.

Virtio Bindings / Linux Bridges

Now let's start to build up the lab starting with the Ethernet connectivity for each vMX as show in Table 4.2. Edit vmx-junosdev.conf as follows:

- Create a link between ge-0/0/1 and ge-0/0/2 on vMX1 for the logical system communication.
- Create a link between ge-0/0/3 on vMX1 and ge-0/0/3 on vMX2.
- Create a bridge for the communication between PE1 on vMX1 interface ge-0/0/0 and the Linux route server VM. You will add the Linux VM to the bridge later.

The configuration is here:

```
interfaces :
-          link_name : vmx_link_ls endpoint_1 :
-          type: junos_dev
vm_name    : vmx1 dev_name  : ge-0/0/1
endpoint_2 :
-          type: junos_dev
vm_name    : vmx1 dev_name  : ge-0/0/2
-          link_name : link_vmx_12 endpoint_1 :
-          type: junos_dev
vm_name    : vmx1 dev_name  : ge-0/0/3
endpoint_2 :
-          type: junos_dev
vm_name    : vmx2 dev_name  : ge-0/0/3

-          link_name : bridge_vmx1_ce1 endpoint_1 :
-          type: junos_dev
vm_name    : vmx1 dev_name  : ge-0/0/0
endpoint_2 :
-          type: bridge_dev dev_name   : bridge_vmx1_ce1
```

Now check, and then apply, the binding configuration. If any of the configuration is already present, but not correct for any reason, the vMX script will fix it. This is useful to know for troubleshooting purposes – if your vMX appears to be fully operational but there is no connectivity, then first check the binding configuration and reapply if necessary:

```
user-1@vmx-day1:~/vmx$ sudo ./vmx.sh --bind-check Checking package ethtool.  [OK]
Check Link vmx_link_ls(ge-0.0.1-vmx1, ge-0.0.2-vmx1) [OK]
Check Link link_vmx_12(ge-0.0.3-vmx1, ge-0.0.3-vmx2) [Not Present]
Check Bridge port bridge_vmx1_ce1(ge-0.0.0-vmx1)..[Not Present]

user-1@vmx-day1:~/vmx$ sudo ./vmx.sh --bind-dev Checking package ethtool.  [OK]
Bind Link vmx_link_ls(ge-0.0.1-vmx1, ge-0.0.2-vmx1)
Warning! Bridge vmx_link_ls already exists [OK]
Bind Link link_vmx_12(ge-0.0.3-vmx1, ge-0.0.3-vmx2) [OK]
```

```
Bind Bridge port bridge_vmx1_ce1(ge-0.0.0-vmx1). [OK]
```

Now let's do a quick test of the LS routers on the vMX1 and then we'll be ready to set up the rest of the lab:

```
user@vmx1> ping logical-system P1 192.168.12.2 rapid
PING 192.168.12.2 (192.168.12.2): 56 data bytes
!!!!!
--- 192.168.12.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss round-trip min/avg/max/
stddev = 1.940/8.880/31.805/11.577 ms

user@vmx1> ping logical-system P1 192.168.15.2 rapid
PING 192.168.15.2 (192.168.15.2): 56 data bytes
!!!!!
--- 192.168.15.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss round-trip min/avg/max/
stddev = 2.179/6.928/22.107/7.629 ms
```

Don't forget you can also configure a logical system and run operational mode commands within the context of the logical system by shifting the CLI to the LS router. Notice the prompt changes to show the name of the LS router:

```
user@vmx1> set cli logical-system P3
logical system: P3

user@vmx1:P3> ping 192.168.35.2 rapid
PING 192.168.35.2 (192.168.35.2): 56 data bytes
!!!!!
--- 192.168.35.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss round-trip min/avg/max/
stddev = 1.694/3.832/9.653/3.088 ms
```

Routing Configuration

OSPF is a popular link-state protocol and enabling it within this lab is a simple process – just configure all the links and loopback addresses within area 0. If you would like to test additional OSPF features such as authentication or reference bandwidth, then go ahead. It's also a good idea to set the router-ID on each router to the loopback IP address.

NOTE It's considered best practice to explicitly disable OSPF on the management interfaces, particularly when using `interface all`. You don't need to do so here, though, because we are using logical systems and the fxp0 is not present in the logical system.

On vMX1:

```
set logical-systems P1 routing-options router-id 10.1.1.1
set logical-systems P1 protocols ospf area 0.0.0.0 interface all interface-type p2p
set logical-systems P1 protocols ospf area 0.0.0.0 interface lo0.1 passive
```

```
set logical-systems P3 routing-options router-id 10.3.3.3
set logical-systems P3 protocols ospf area 0.0.0.0 interface all interface-type p2p
set logical-systems P3 protocols ospf area 0.0.0.0 interface lo0.3 passive
set logical-systems PE1 routing-options router-id 10.5.5.5
set logical-systems PE1 protocols ospf area 0.0.0.0 interface all interface-type p2p
set logical-systems PE1 protocols ospf area 0.0.0.0 interface lo0.5 passive
```

On vMX2:

```
set logical-systems P2 routing-options router-id 10.2.2.2
set logical-systems P2 protocols ospf area 0.0.0.0 interface all interface-type p2p
set logical-systems P2 protocols ospf area 0.0.0.0 interface lo0.2 passive
set logical-systems P4 routing-options router-id 10.4.4.4
set logical-systems P4 protocols ospf area 0.0.0.0 interface all interface-type p2p
set logical-systems P4 protocols ospf area 0.0.0.0 interface lo0.4 passive
set logical-systems PE2 routing-options router-id 10.6.6.6
set logical-systems PE2 protocols ospf area 0.0.0.0 interface all interface-type p2p
set logical-systems PE2 protocols ospf area 0.0.0.0 interface lo0.6 passive
```

Don't forget to verify that the OSPF neighbors are fully established. Each PE router should have two neighbors and each P router should have three neighbors:

```
user@vmx1> show ospf neighbor logical-system P1
Address    Interface  State  IDPri Dead
192.168.13.2 ge-0/0/1.13  Full 10.3.3.3 128  34
192.168.15.2 ge-0/0/1.15  Full 10.5.5.5 128  34
192.168.12.2 ge-0/0/3.12  Full 10.2.2.2 128  36

user@vmx1> show ospf neighbor logical-system P3
Address    Interface  State  IDPri Dead
192.168.35.2 ge-0/0/1.35  Full 10.5.5.5 128  34
192.168.13.1 ge-0/0/2.13  Full 10.1.1.1 128  33
192.168.34.2 ge-0/0/3.34  Full 10.4.4.4 128  32

user@vmx1> show ospf neighbor logical-system PE1
Address    Interface  State  IDPri Dead
192.168.15.1 ge-0/0/2.15  Full 10.1.1.1 128  34
192.168.35.1 ge-0/0/2.35  Full 10.3.3.3 128  36

user@vmx2> show ospf neighbor logical-system P2
Address    Interface  State  IDPri Dead
192.168.12.1 ge-0/0/3.12  Full 10.1.1.1 128  38
192.168.24.2 lt-0/0/0.242 Full 10.4.4.4 128  39
192.168.26.2 lt-0/0/0.262 Full 10.6.6.6 128  30

user@vmx2> show ospf neighbor logical-system P4
Address    Interface  State  IDPri Dead
192.168.34.1 ge-0/0/3.34  Full 10.3.3.3 128  39
192.168.24.1 lt-0/0/0.244 Full 10.2.2.2 128  33
192.168.46.2 lt-0/0/0.464 Full 10.6.6.6 128  39

user@vmx2> show ospf neighbor logical-system PE2
Address    Interface  State  IDPri Dead
192.168.26.1 lt-0/0/0.266 Full 10.2.2.2 128  35
192.168.46.1 lt-0/0/0.466 Full 10.4.4.4 128  35
```

You could also run the `show ospf neighbor logical-system all` command to show the neighbors for all logical system routers with just one CLI command. Spend a few moments here checking routing tables and running ping tests to be sure that each router has full reachability to the rest of the network.

MPLS Configuration

You have already enabled the MPLS family on the router-to-router interfaces, so all that needs to be done here is to enable the RSVP and MPLS protocols on each router and to set up an LSP between PE1 and PE2. Specify the interfaces individually, if you prefer.

On vMX1:

```
set logical-systems P1 protocols rsvp interface all
set logical-systems P1 protocols mpls interface all
set logical-systems P3 protocols rsvp interface all
set logical-systems P3 protocols mpls interface all
set logical-systems PE1 protocols rsvp interface all
set logical-systems PE1 protocols mpls interface all
```

On vMX2:

```
set logical-systems P2 protocols rsvp interface all
set logical-systems P2 protocols mpls interface all
set logical-systems P4 protocols rsvp interface all
set logical-systems P4 protocols mpls interface all
set logical-systems PE2 protocols rsvp interface all
set logical-systems PE2 protocols mpls interface all
```

Now let's build the LSP between PE1 and PE2. Remember that an LSP is unidirectional, so the configuration must be applied on both PE1 and PE2.

On vMX1:

```
set logical-systems PE1 protocols mpls label-switched-path to-PE2 to 10.6.6.6
set logical-systems PE1 protocols mpls label-switched-path to-PE2 no-cspf
```

On vMX2:

```
set logical-systems PE2 protocols mpls label-switched-path to-PE1 to 10.5.5.5
set logical-systems PE2 protocols mpls label-switched-path to-PE1 no-cspf
```

Make sure that you verify that the LSP has established correctly. If for some reason the LSP is down, then use the `extensive` option to look for the reason. Perhaps the destination is missing from the route table, or MPLS is not enabled on an interface (family MPLS or protocol MPLS):

```
user@vmx2> show mpls lsp logical-system PE2
Ingress LSP: 1 sessions
To         From State Rt P ActivePath LSPname
10.5.5.5   10.6.6.6 Up 0 * to-PE1 Total 1 displayed, Up 1, Down 0

Egress LSP: 1 sessions
To         From State  Rt Style Labelin Labelout LSPname
10.6.6.6   10.5.5.5 Up 0 1 FF 3 - to-PE2 Total 1 displayed, Up 1, Down 0

Transit LSP: 0 sessions
Total 0 displayed, Up 0, Down 0
```

Adding Another Non-vMX Virtual Machine

Now let's build a Linux VM and configure it to be a BGP route server using *ExaBGP* software. A useful addition to any lab!

ExaBGP is an application that facilitates an easy way to interact with BGP networks. See https://github.com/Exa-Networks/exabgp for details.

To build a Linux VM with virsh:

1. Download Ubuntu directly to your home directory on the KVM host:

```
user-1@vmx-day1:~/vmx$ cd

user-1@vmx-day1:~$ wget http://archive.ubuntu.com/ubuntu/dists/trusty/main/installer-amd64/ current/
images/netboot/mini.iso
```

2. You may need install the "virtinst" package. This is a CLI tool to create a new VM:

```
user-1@vmx-day1:~$ sudo apt-get install virtinst
```

3. Create a directory to store your VM disk images:

```
user-1@vmx-day1:~$ mkdir VMs
```

4. Build the VM using the virt-install tool. This tool will create the VM configuration files and spawn a VNC server for you to connect to:

```
user-1@vmx-day1:~$ sudo virt-install --virt-type kvm --name Linux-day1 --ram 1024 --cdrom Linux.
iso  --disk VMs/linux-day1.img,size=2 --network bridge=br-ext --network bridge=bridge_vmx1_ce1 --os-
type=linux --os-variant=ubuntu16.04 --graphics vnc,listen=0.0.0.0,port=5910 -noautoconsole
```

Most of these command-line options are self-explanatory but there are two of note:

- Network configuration – there are two interfaces added to the host so that you can SSH to the VM. The eth0 interface connects to the management interface bridge (br-ext) also used by the vMX, and the second interface connects to the PE1_CE1 bridge that was created earlier.
- Graphics configuration – VNC is used to complete the installation and is configured here to listen on port 5910.

5. Load up your VNC client and connect to the IP address of the KVM host on port 5910. The VNC window will take you to the Ubuntu installer screen. Select *ens192* as the primary interface and then progress through the installer as you did in Chapter 2.
6. After a short wait the installation will complete and the system reboots.

Connect to Linux VM

After installation, check that the new VM is running. This is done using `virsh`. Notice the `--all` option to show domains that are not running. If the `--all` is not specified, then only information on running domains is shown:

```
user-1@vmx-day1:~$ sudo virsh list --all
Id         Name State
2          vcp-vmx1 running
3          vfp-vmx1 running
5          vcp-vmx2 running
6          vfp-vmx2 running
-          linux-day1 shut off
```

Here you can see the vMX VMs are running but the new Linux VM linux-day1 is shut off. It's time to start it up:

```
user-1@vmx-day1:~$ sudo virsh start linux-day1
Domain linux-day1 started

user-1@vmx-day1:~$ sudo virsh list --all
Id         Name State
2          vcp-vmx1 running
3          vfp-vmx1 running
5          vcp-vmx2 running
6          vfp-vmx2 running
20         linux-day1 running
```

You can now connect back to the VM with VNC. If you prefer to SSH directly in to the VM, connect to the CLI with VNC and `install openssh-server`:

```
user-1@linux-day1: ~$ sudo apt-get install openssh-server
```

Once the SSH service has been installed you can SSH to the IP address assigned to the VM interface ens192. This interface will be using DHCP unless you prefer to change the configuration to be statically assigned. The `ifconfig eth0` command can be used to find out the IP address to use for SSH. To complete the build of this Linux CE1 it's necessary to configure the CE1-PE1 interface. Use the IP addressing as shown in Table 4.2.

Connect to CE1 via SSH or VNC and modify the configuration of eth1:

```
user-1@linux-day1:~$ cd /etc/network/
user-1@linux-day1:/etc/network$ sudo vi interfaces
```

Update the interfaces configuration:

```
auto ens192
iface eth1 inet static address 10.0.0.1
network 255.255.255.0
```

And bring up the interface:

```
user-1@linux-day1:/etc/network$ sudo ifup ens192
```

The build of CE1 is now complete!

Check the VM to vMX Bridges

The installer has automatically connected the VM interfaces to the correct bridges. If you need to check that everything is correct or to make changes manually, this can be done with the following steps.

Use `virsh` to show you which interfaces on the VM are mapped to the Linux Bridges. Here you can see vnet0 is assigned to br-ext and vnet1 to bridge_vmx1_ce1:

```
user-1@vmx-day1:~$ sudo virsh domiflist linux-day1
Interface Type Source Model MAC
vnet0      bridge br-ext virtio 52:54:00:bb:a4:77
vnet1      bridge bridge_vmx1_ce1 virtio  52:54:00:69:99:e2
```

You can see the same thing by looking at the bridges directly:

```
user-1@vmx-day1:~$ brctl show br-ext
bridge name  bridge id  STP enabled  interfaces
br-ext     8000.000c29510c44yes  br-ext-nic eth0 vcp_ext-

vmx1 vmx2 vmx1 vmx2


vcp_ext- vfp_ext- vfp_ext- vnet0

user-1@vmx-day1:~$ brctl show bridge_vmx1_ce1
bridge name  bridge id  STP enabled  interfaces bridge_vmx1_ce1 8000.fe060a0efff0   no     ge-0.0.0-
vmx1
vnet1
```

As expected, the correct VM interface was assigned to each bridge. If the VM interfaces were assigned to the default bridge, they can easily be corrected by deleting the interface from the bridge, and then reassigning it to a new bridge, like this:

```
user-1@vmx-day1:~$ sudo brctl delif virbr0 vnet1

user-1@vmx-day1:~$ sudo brctl addif bridge_vmx1_ce1 vnet1
```

Putting It All Together

A Virtual Private LAN service (VPLS) will appear to connect CE devices as an Ethernet LAN. This is accomplished by the VPLS incorporating LAN-like functionality such as MAC learning, flooding, and forwarding across an MPLS network.

VPLS Configuration

VPLS is defined in two different RFCs – RCF4761 (BGP Auto-Discovery and Signaling for VPLS) and RFC4762 (Virtual Private LAN Service over LDP). In this lab you will be configuring LDP-signaled VPLS.

To complete the VPLS configuration, as you are using LDP signaled VPLS, LDP must be enabled on the loopback interface of each PE. The VPLS configuration itself is done within a routing instance.

NOTE There is no auto-discovery with LDP-signaled VPLS, so when you configure the PE routers you need to specify every neighbor PE that is participating in the VPLS. This static neighbor configuration with LDP signaled VPLS is one reason why BGP signaled VPLS scales better in a large network, but LDP signaled is fine for this lab.

1. Enable LDP on the PE loopback interfaces. Remember PE1 is running on vMX1 and PE2 is running on vMX2:

```
set logical-systems PE1 protocols ldp interface lo0.5
set logical-systems PE2 protocols ldp interface lo0.6
```

2. Create router CE2 as a logical system router on vMX2 and add the point-to-point link between CE2 and PE2:

```
set logical-systems CE2 interfaces lt-0/0/0 unit 688 encapsulation ethernet
set logical-systems CE2 interfaces lt-0/0/0 unit 688 peer-unit 686
set logical-systems CE2 interfaces lt-0/0/0 unit 688 family inet address 10.0.0.2/24
set logical-systems PE2 interfaces lt-0/0/0 unit 686 encapsulation ethernet-vpls
set logical-systems PE2 interfaces lt-0/0/0 unit 686 peer-unit 688
```

3. Configure the VPLS routing instances. On PE1 the VPLS interface is the Ethernet interface facing CE1 (ge-0/0/0). Since you are only using VPLS on the interface, it's okay to use an encapsulation of Ethernet-VPLS rather than flexible-ethernet-services:

On vMX1:

```
set interfaces ge-0/0/0 encapsulation ethernet-vpls
set logical-systems PE1 interfaces ge-0/0/0 unit 0
set logical-systems PE1 routing-instances VPLS instance-type vpls
set logical-systems PE1 routing-instances VPLS interface ge-0/0/0.0
set logical-systems PE1 routing-instances VPLS protocols vpls vpls-id 1
set logical-systems PE1 routing-instances VPLS protocols vpls neighbor 10.6.6.6
```

4. Create the VPLS on PE2. This time the VPLS interface isn't a Gigabit interface, it is the `lt` interface on PE2 that connects to CE2, so be sure to set the encapsulation to `ethernet-vpls` rather than `ethernet`. Being able to use a lt interface with VPLS is another reason that they are so flexible!

On vMX2:

```
set logical-systems PE2 interfaces lt-0/0/0 unit 686 encapsulation ethernet-vpls
set logical-systems PE2 routing-instances VPLS instance-type vpls
set logical-systems PE2 routing-instances VPLS interface lt-0/0/0.686
set logical-systems PE2 routing-instances VPLS protocols vpls vpls-id 1
set logical-systems PE2 routing-instances VPLS protocols vpls neighbor 10.5.5.5
```

5. Verification – check that VPLS has been established on PE1 and PE2:

```
user@vmx2> show vpls connections logical-system PE2
Layer-2 VPN connections: Legend for connection status (St)
EI -- encapsulation invalid NC -- interface encapsulation not CCC/TCC/VPLS
EM -- encapsulation mismatch  WE -- interface and instance encaps not same VC-Dn -- Virtual circuit
```

```
down       NP -- interface hardware not present
CM -- control-word mismatch    -> -- only outbound connection is up CN -- circuit not provisioned
<- -- only inbound connection is up OR -- out of range  Up -- operational
OL -- no outgoing label  Dn -- down
LD -- local site signaled down CF -- call admission control failure
RD -- remote site signaled down SC -- local and remote site ID collision LN -- local site not designated
LM -- local site ID not minimum designated
RN -- remote site not designated RM -- remote site ID not minimum designated XX -- unknown connection
status IL -- no incoming label
MM -- MTU mismatch  MI -- Mesh-Group ID not available BK -- Backup connection   ST -- Standby connection
PF -- Profile parse failure    PB -- Profile busy RS -- remote site standby SN -- Static Neighbor
LB -- Local site not best-site RB -- Remote site not best-site VM -- VLAN ID mismatch
Legend for interface status
Up -- operational
Dn -- down
Instance: VPLS
VPLS-id: 1
Neighbor                  Type  St   Time last up          # Up trans
10.5.5.5(vpls-id 1)   rmt    Up   Feb 23 16:21:22 2016    1
Remote PE: 10.5.5.5, Negotiated control-word: No
Incoming label: 800000, Outgoing label: 800000 Negotiated PW status TLV: No
Local interface: vt-0/0/10.51380224, Status: Up, Encapsulation: ETHERNET
Description: Intf - vpls VPLS neighbor 10.5.5.5 vpls-id 1
Flow Label Transmit: No, Flow Label Receive: No



user@vmx1> show vpls connections logical-system PE1
Layer-2 VPN connections: Legend for connection status (St)
EI -- encapsulation invalid     NC -- interface encapsulation not CCC/TCC/VPLS
EM -- encapsulation mismatch      WE -- interface and instance encaps not same
VC-Dn -- Virtual circuit down   NP -- interface hardware not present
CM -- control-word mismatch     -> -- only outbound connection is up
CN -- circuit not provisioned   < only inbound connection is up
OR -- out of range                Up -- operational
OL -- no outgoing label         Dn -- down
LD -- local site signaled down    CF -- call admission control failure
RD -- remote site signaled down SC -- local and remote site ID collision
LN -- local site not designated LM -- local site ID not minimum designated
RN -- remote site not designated RM -- remote site ID not minimum designated
XX -- unknown connection status IL -- no incoming label
MM -- MTU mismatch                MI -- Mesh-Group ID not available
BK -- Backup connection         ST -- Standby connection
PF -- Profile parse failure     PB -- Profile busy
RS -- remote site standby       SN -- Static Neighbor
LB -- Local site not best-site    RB -- Remote site not best-site
VM -- VLAN ID mismatch

Legend for interface status
Up -- operational
Dn -- down
Instance: VPLS
VPLS-id: 1
Neighbor   Type St  Time last up # Up trans 10.6.6.6(vpls-id 1)    rmt NP
```

Here you can see that the VPLS is up on PE2, but PE1 is showing an error `interface hardware not present`. Remember that PE2 is using lt interfaces and so has been configured with `tunnel-ser- vices`, but PE1 has not. VPLS requires that tunnel services

be configured, or for a router without tunnel services the `no-tunnel-services` statement will create a label-switched interface (LSI) to enable the VPLS functionality to work. Let's use `no-tunnel-services` so you can see the difference. Add the following to vMX1:

```
user@vmx1> set logical-systems PE1 routing-instances VPLS protocols vpls no-tunnel-services

user@vmx1> show vpls connections logical-system PE1

Layer-2 VPN connections:
Legend for connection status (St)
EI -- encapsulation invalid   NC -- interface encapsulation not CCC/TCC/VPLS
EM -- encapsulation mismatch    WE -- interface and instance encaps not same
VC-Dn -- Virtual circuit down   NP -- interface hardware not present
CM -- control-word mismatch     -> -- only outbound connection is up
CN -- circuit not provisioned   <- -- only inbound connection is up
 OR -- out of range             Up -- operational
OL -- no outgoing label         Dn -- down
LD -- local site signaled down  CF -- call admission control failure
RD -- remote site signaled down SC -- local and remote site ID collision
LN -- local site not designated LM -- local site ID not minimum designated
RN -- remote site not designatedRM -- remote site ID not minimum designated
XX -- unknown connection status IL -- no incoming label
MM -- MTU mismatch                MI -- Mesh-Group ID not available
BK -- Backup connection         ST -- Standby connection
PF -- Profile parse failure     PB -- Profile busy RS -- remote site standby
SN -- Static Neighbor
LB -- Local site not best-site   RB -- Remote site not best-site
VM -- VLAN ID mismatch
Legend for interface status
Up -- operational
Dn -- down
Instance: VPLS
VPLS-id: 1
Neighbor                  Type  St   Time last up          # Up trans
10.6.6.6(vpls-id 1)   rmt    Up   Feb 23 16:27:44 2016    1
Remote PE: 10.6.6.6, Negotiated control-
word: No Incoming label: 262145, Outgoing label: 800000 Negotiated PW status TLV: No
Local interface: lsi.17825792, Status: Up, Encapsulation: ETHERNET
Description: Intf - vpls VPLS neighbor 10.6.6.6 vpls-id 1
Flow Label Transmit: No, Flow Label Receive: No
```

Great. The VPLS on PE1 is now up. Notice the LSI local interface on PE1 compared with the VT local interface on PE2.

At this point the VPLS configuration is complete, and you should have direct connectivity between CE1 and CE2. Now let's configure the BGP peering between CE1 and CE2. Let's first check that everything is okay using a quick ping across the VPLS:

```
user@vmx2> ping logical-system CE2 10.0.0.1 rapid
PING 10.0.0.1 (10.0.0.1): 56 data bytes
!!!!!
--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss round-trip min/avg/max/stddev =
2.798/3.071/3.536/0.269 ms
```

```
user-1@linux-day1:/etc/network$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=2.64 ms 64 bytes from 10.0.0.2: icmp_
seq=2 ttl=64 time=1.79 ms
^C
--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms

rtt min/avg/max/mdev = 1.790/2.216/2.642/0.426 ms
```

Try It Yourself: BGP Signaled VPLS

See what you can learn about BGP and VPLS by configuring BGP signaled VPLS instead of LDP signaled VPLS:

- *Configure BGP between PE1 and PE2. Enable the L2VPN signaling address family.*
- *Create VPLS routing instances and assign CE interfaces. Verify CE connectivity.*

NOTE EVPN is a combined Layer 2 and Layer 3 VPN solution that is more scalable, resilient, and efficient than current technologies. It provides several benefits including greater network efficiency, reliability, scalability, virtual machine (VM) mobility, and policy control for service providers and enterprises. For more details on migrating to EVPN on account of the scaling benefits and ease of deployment, see https://www.juniper.net/documentation/us/en/software/junos/evpn-vxlan/topics/concept/bgp-vpls-to-evpn-migration.html.

Route Server Configuration

The final step in this *Day One* book is to configure a BGP peering between CE1 and CE2. The reason for using a Linux server running ExaBGP is because it provides you with flexibility in BGP announcements.

In this book's case, CE2 will simulate a peering router; in other words, it will be configured to advertise eBGP routing information, and iBGP will be configured between CE1 and CE2.

For the route server, use the ExaBGP software. It is available from https://github.com/Exa-Networks/exabgp.

MORE? Refer to a very useful resource for deploying route servers in an Internet Exchange Point (IXP) at https://www.juniper.net/documentation/en_US/day-one-books/DO_JunosRouteServersUPDATE.pdf.

Connect to the CE VM and use `wget` to download ExaBGP:

```
user-1@linux-day1:~$ wget https://github.com/Exa-Networks/exabgp/archive/3.4.10.tar.gz
```

Extract ExaBGP and create a simple configuration file called day1.conf:

```
user-1@linux-day1:~$ tar -xzf 3.4.10.tar.gz
user-1@linux-day1:~$ cd exabgp-3.4.10/
user-1@linux-day1:~/exabgp-3.4.10$
user-1@linux-day1:~/exabgp-3.4.10$ vi day1.conf
```

Set up the configuration file as shown next. The options should be self-explanatory. It's a very simple file, but as you can see the tool is quite flexible. There is also an API to allow more advanced route advertisement functionality. ExaBGP is a powerful tool, and this configuration is just a subset of what is possible:

```
 group internal {
           hold-time 180;
           local-as 65000;
           peer-as 65000;
           router-id 10.0.0.1;
           static {
             route 10.10.10.0/24 next-hop 10.0.0.1 as-path [ 10 ] ;
             route 10.10.20.0/24 next-hop 10.0.0.1 as-path [ 10 20 ] ;
             route 10.10.30.0/24 next-hop 10.0.0.1 as-path [ 10 20 30 ] ;
             route 10.10.40.0/24 next-hop 10.0.0.1 as-path [ 40 50 60 ] ;
             route 10.10.50.0/24 next-hop 10.0.0.1 as-path [ 70 80 90 ] ;
             route 10.10.60.0/24 next-hop 10.0.0.1 as-path [ 100 120 130 ] ;
             route 10.10.70.0/24 next-hop 10.0.0.1 as-path [ 140 150 160 ] ;
             route 10.10.80.0/24 next-hop 10.0.0.1 as-path [ 170 180 ] ;
             }
           neighbor 10.0.0.2 {
             local-address 10.0.0.1;
             }
           }
```

Start ExaBGP with this command – it will run in the foreground:

```
user-1@linux-day1:~/exabgp-3.4.10$ sbin/exabgp day1.conf
```

Finally, configure a BGP neighbor on CE2:

```
set logical-systems CE2 protocols bgp local-as 65000
set logical-systems CE2 protocols bgp group internal type internal
set logical-systems CE2 protocols bgp group internal peer-as 65000
set logical-systems CE2 protocols bgp group internal neighbor 10.0.0.1
```

With ExaBGP running, the session will come up, and you will see the BGP routes being received:

```
user@vmx2> show route logical-system CE2

inet.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
10.0.0.0/24
*[Direct/0] 01:44:38
```

```
> via lt-0/0/0.688 10.0.0.2/32  *[Local/0] 01:44:38
Local via lt-0/0/0.688 10.10.10.0/24
*[BGP/170] 00:01:07, localpref 100
AS path: 10 I, validation-state: unverified
> to 10.0.0.1 via lt-0/0/0.688 10.10.20.0/24
*[BGP/170] 00:01:07, localpref 100
AS path: 10 20 I, validation-state: unverified
> to 10.0.0.1 via lt-0/0/0.688 10.10.30.0/24
*[BGP/170] 00:01:07, localpref 100
AS path: 10 20 30 I, validation-state: unverified
> to 10.0.0.1 via lt-0/0/0.688 10.10.40.0/24
*[BGP/170] 00:01:07, localpref 100
AS path: 40 50 60 I, validation-state: unverified
> to 10.0.0.1 via lt-0/0/0.688 10.10.50.0/24
*[BGP/170] 00:01:07, localpref 100
AS path: 70 80 90 I, validation-state: unverified
> to 10.0.0.1 via lt-0/0/0.688 10.10.60.0/24
*[BGP/170] 00:01:07, localpref 100
AS path: 100 120 130 I, validation-state: unverified
> to 10.0.0.1 via lt-0/0/0.688 10.10.70.0/24
*[BGP/170] 00:01:07, localpref 100
AS path: 140 150 160 I, validation-state: unverified
> to 10.0.0.1 via lt-0/0/0.688 10.10.80.0/24
*[BGP/170] 00:01:07, localpref 100
AS path: 170 180 I, validation-state: unverified
> to 10.0.0.1 via lt-0/0/0.688
```

At this point you have a working VPLS environment and BGP running directly across the VPLS between the two CE devices. You can further modify the BGP configuration on CE2 to include BGP policy, and adjust the route announcements on CE1 to validate your policy is working correctly.

Try It Yourself: Expand the Environment Further

You have now built a flexible lab environment – why not try to expand it further? Add another PE router device to your lab and add a third CE router to the VPLS. Try to configure MPLS VPNs rather than VPLS, and add the three CE routers to the VPN. You are only limited by your imagination!

Summary

This chapter showed you how simple it is to scale your lab environment to a multi-router topology with multiple vMXs and logical systems. Large lab topologies can easily be created and the powerful feature set of vMX allows you to test many different protocols and topologies. Now that you've built this environment – expand it further on your own and have fun learning!

Chapter 5

Troubleshooting

This chapter highlights a few more ways to troubleshoot vMX operation. Generally speaking, you can accomplish most things on vMX with Juniper's vmx.sh orchestration script, but just in case things are not going entirely to plan, here are a few more troubleshooting tips for you to try out.

Verify vMX VM State

To verify the VMs you can use libvirt's `virsh list` command. This will display the name and state of the vMX VMs on your KVM hosts. The state can be running, idle, paused, shut down, crashed, or dying.

Use the following commands to start and stop VMs, but ideally you will use the vmx.sh script to manage this process:

- `virsh destroy`—Force stops a VM (but does not delete the VM).
- `virsh start`—Starts an inactive VM.

VCP and VFP Communication

If you run the `show interfaces terse` command on the VCP and you do not see any ge-0/0/x interfaces listed, then it is possible that the connection between the VCP and VFP has not established or that the VFP has not booted up correctly.

On your first attempt to log in to the VFP console, if you do not get a response (i.e., the console appears to have locked) then it is possible that the VFP has not booted up.

To check for VFP errors during boot up:

Start the vMX using the orchestration script. As soon as the VFP boots, be ready to console in to the VFP:

```
./vmx.sh –console vfp vmx1
```

Look for any error messages or kernel panic during boot. For example, if you see *Kernel panic - not syncing: Attempted to kill init! exitcode=0x0000000b* then go and check that your BIOS settings are correct.

If VFP has booted correctly and you see a log in prompt on the console (log in with the default or root username / password), look for syslog entries containing the RPIO or HOSTIF messages:

```
RPIO: Accepted connection from 172.16.0.1:50896 <–> vPFE:3000 RPIO: Accepted connection from
172.16.0.1:56098 <–> vPFE:3000 HOSTIF: Accepted connection
```

If the VCP cannot connect to the VFP, and the VFP syslog file does not display the RPIO and HOSTIF messages, you should follow the next few procedures:

1. Run the `request chassis fpc slot 0 restart` command from the VCP CLI. If an FPC is in transition, and an error message is displayed, then run `restart chassis–control`.
2. If these commands do not correct the problem, check to see if you can ping the VFP from the VCP routing-instance `_ _ juniper _ private1 _ _` via the internal bridge:

```
user> ping 128.0.0.16 routing–instance juniper_private1 PING 128.0.0.16 (128.0.0.16): 56 data bytes
64 bytes from 128.0.0.16: icmp_seq=0 ttl=64 time=0.273 ms 64 bytes from 128.0.0.16: icmp_seq=1 ttl=64
time=0.606 ms
```

NOTE The IP addresses that are automatically assigned to the em1 interface on the VCP and internal interface on the VFP in 20.2R2 are: VCP 128.0.0.1, VFP 128.0.0.16.

If you still have problems, then perform these additional steps:

1. Check that the Linux bridge configuration is correct. Run `brctl show` from the KVM host shell and check the internal bridge configuration.
2. Using `virsh`, check that both VMs are actually running. Run `sudo virsh –c qemu:///system list`.
3. Restart the FPC from the VCP VM. Run `request chassis fpc restart`.
4. Restart the chassis management process from VCP VM. Run `restart chassis–control`.
5. Stop and start the VFP VM.

6. Stop and start the VCP VM.
7. Restart the KVM host.

If completing all of the above steps has not helped, then it's time for you to talk to JTAC or post on Juniper's Elevate forums.

VFP Log Files

If you wish to look at the forwarding plane log files, they can be found as log files, and for Linux these are usually located in /var/log. Crash logs can be found in /var/crash.

Virtio Troubleshooting

If the VCP and VFP are operational but you have lost connectivity between vMX instances on the same host or from a vMX instance to external devices, then the first place to start looking (if you are using virtio interfaces) is the Linux bridges on the KVM host. You can check these out yourself or use the vmx.sh script to check for you.

To check the Linux bridges yourself, `run brctl show` on the host and check that the correct bridges are present and that all the interfaces you expect to see have been added to the bridges. Remember the interfaces could be a combination of vMX instance virtual interfaces and KVM host physical interfaces.

If you are using the vmx.sh script to check the bridges, then follow this troubleshooting procedure. If you want to see a working set of bridges:

```
user@vmx1:~/vmx$ brctl show
bridge name     bridge id           STP enabled     interfaces
br-ext          8000.0050569363d4       yes             br-ext-nic
                                                        ens192
                                                        vcp-ext-vmx1
                                                        vcp-ext-vmx2
                                                        vfp-ext-vmx1
                                                        vfp-ext-vmx2
br-int-vmx1     8000.525400f80290       yes             br-int-vmx1-nic
                                                        vcp-int-vmx1
                                                        vfp-int-vmx1
br-int-vmx2     8000.525400e70698       yes             br-int-vmx2-nic
                                                        vcp-int-vmx2
                                                        vfp-int-vmx2
bridge_vm12     8000.fe060a0efff0       no              ge-0.0.0-vmx1
                                                        ge-0.0.0-vmx2
virbr0          8000.5254003ee6a1       yes             virbr0-nic
```

If you run `./vmx.sh --bind-check` the script will tell you if everything is configured correctly:

```
lab@ix-ubuntu-02:~/vmx$ sudo ./vmx.sh --bind-check
Checking package ethtool.........................[OK]
Check Bridge port bridge_vm12(ge-0.0.0-vmx1)......[Not Present]
Check Bridge port bridge_vm12(ge-0.0.0-vmx2)......[Not Present]
```

In the above output, bridges are not present. Run `./vmx.sh --bind-dev` to bind the interfaces:

```
lab@ix-ubuntu-02:~/vmx$ sudo ./vmx.sh --bind-dev
Checking package ethtool.........................[OK]
Bind Bridge port bridge_vm12(ge-0.0.0-vmx1).......[OK]
Bind Bridge port bridge_vm12(ge-0.0.0-vmx2).......[OK]
```

Now run `./vmx.sh --bind-check` again to see if the bridges are present:

```
lab@ix-ubuntu-02:~/vmx$ sudo ./vmx.sh --bind-check
Checking package ethtool.........................[OK]
Check Bridge port bridge_vm12(ge-0.0.0-vmx1)......[OK]
Check Bridge port bridge_vm12(ge-0.0.0-vmx2)......[OK]
```

As you can see here, everything looks good. Now what would happen if you were to delete one of the vMX interfaces from a bridge:

```
user@vmx1:~/vmx$ sudo brctl delif bridge_vmx12 ge-0.0.0-vmx2
user@vmx1:~/vmx$ sudo ./vmx.sh --bind-check

Checking package ethtool.........................[OK]
Check Bridge port bridge_vm12(ge-0.0.0-vmx1)......[OK]
Check Bridge port bridge_vm12(ge-0.0.0-vmx2)......[Misconfigured]
```

You can see how the script has reported back that the bridge is misconfigured because ge-0/0/0 on vMX2 is missing from the bridge. This is easily fixed by re-running vmx.sh with the `--bind-dev` option:

```
lab@ix-ubuntu-02:~/vmx$ sudo ./vmx.sh --bind-dev
Checking package ethtool.........................[OK]
Bind Bridge port bridge_vm12(ge-0.0.0-vmx1).......[OK]
Bind Bridge port bridge_vm12(ge-0.0.0-vmx2).......[OK]
```

As you can see, the vMX script has corrected the error for you.

If you receive an error like the one shown here during a vMX start operation, and the vMX will not start, it is probably due to the the vMX VMs not being stopped by the orchestration script (perhaps the physical host was rebooted without the vMX being shut down):

```
================================================
System Setup Completed
================================================
Generate libvirt files. [OK]
Sleep 2 secs.[OK]
Find configured management interface.  eth0
Find existing management gateway.  br-ext
```

```
Check if eth0 is already enslaved to br-ext.[Yes]
Create br-int-vmx2. [Failed]
error: Failed to define network from /home/user/vmx/build/vmx2/xml/br-int- generated.xml
error: operation failed: network 'br-int-vmx2' already exists with uuid 96aa825b-5f02-48ca- bbb4-
135b6a7e89ce
Log file.  /dev/null
================================================
Aborted!. 1 error(s) and 0 warning(s)
================================================
```

To correct this issue, you can run the orchestration script `start` again. Don't run the `--cleanup` option because this will clean up all information about the vMX instance including your Junos configuration!

Chapter 6

Getting Started with vMX on VMware

Although this book has focused on the KVM release of vMX, there is also a VMware release available. This chapter shows you how to install vMX on VMware.

Once you have installed vMX then be sure to walk through all of the lab exercises shown here on your VMware build of vMX.

vMX Installation

If you have a valid login, you can download vMX directly from the vMX download page: https://support.juniper.net/support/downloads/?p=vmx#sw.

Once you have downloaded the vMX software, load up the client for your ESXi server and log in. This chapter uses ESXi 6.5 and vMX version 20.2R2 for the demonstrations.

ALERT! Be sure to check the latest install and operating guides for your current version of vMX. It is a vibrant technology and Juniper is making improvements to vMX every calendar quarter. At the time of this writing, the process to install vMX on VMware is as written here.

vMX Bundle Details

Before progressing any further you will need to extract the vMX package. The content of the VMware package is as shown here:

- ova/vcp_*.ova: This is the software image file for VCP.
- ova/vfpc_*.ova: And this is the software image file for VFP.

Set Up the vMX Network

The VMware release is no different than the KVM release when it comes to the required default networks. You must configure a minimum of three networks:

- Management network (br-ext)
- Internal network for VCP and VFP communication (br-int)
- Data interfaces

To create these networks, log in to the ESXi Web console. In the left navigation pane, select Networking as shown in Figure 6.1. In the top right panel select the Virtual switches tab and then click Add standard virtual switch option.

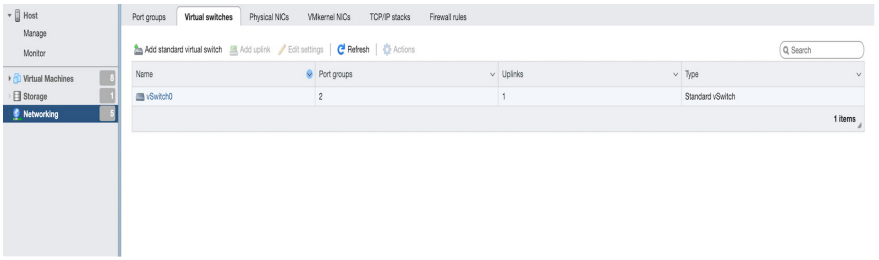

Figure 6.1 *Add Virtual Switch*

Now let's create each of the required three networks.

Management Network

Use the following steps to create the management network:

1. Select Networking in the left-hand pane and go to the Port groups tab shown in Figure 6.2.

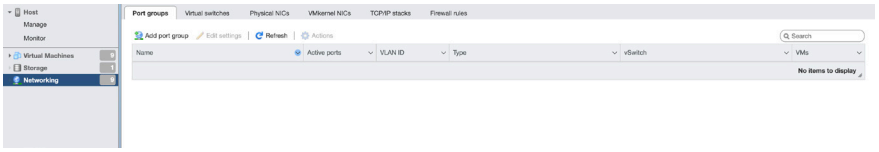

Figure 6.2 *Management Network Step 1*

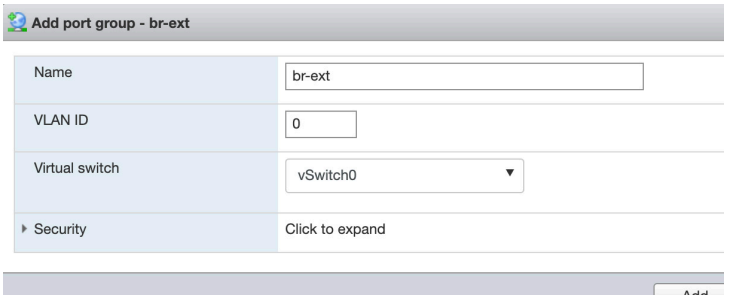

Figure 6.3 *Management Network Step 2*

3. Under the Security tab, enable Promiscuous mode, MAC address changes, and Forged transmits and shown in Figure 6.4.

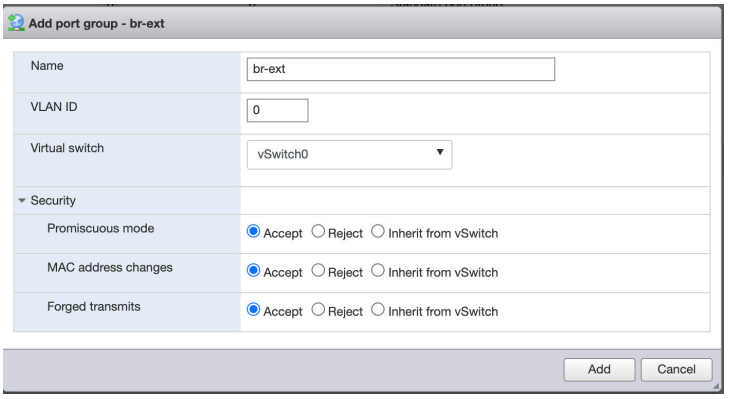

Figure 6.4 *Management Network Step 3*

4. Click Add. You will see the new port group "br-ext" has been added to the standard switch vSwitch0 as shown in Figure 6.5.

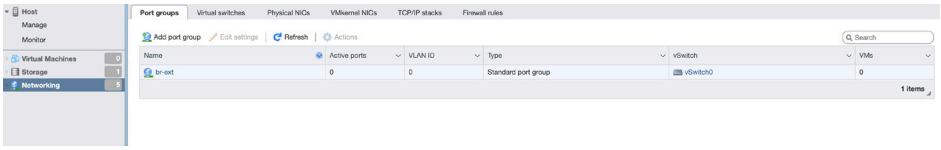

Figure 6.5 *Management Network Step 4*

This network is now connected to the management port (for example, vmnic0), which has a management IP address.

Internal Network

The internal network is used only for communication between the VCP and VFP. A separate internal network is required for each vMX instance.

1. Within Networking, go to Virtual switches tab.
2. Select Add standard virtual switch-br-ext-vswitch, as shown in Figure 6.6. Do not add any uplink to the vSwitch.

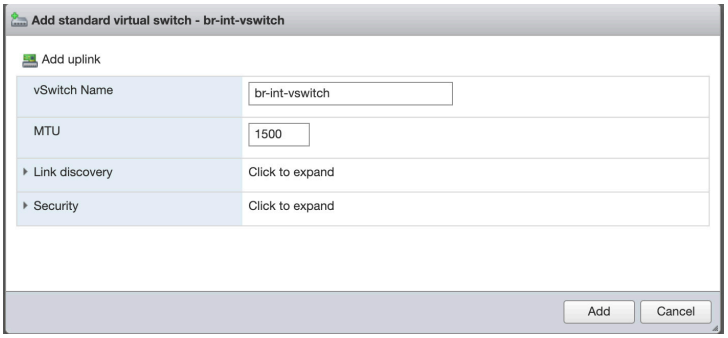

Figure 6.6 Internal Network Step 1&2

3. Enable all security options: Promiscuous mode, MAC address changes, and Forged transmits as shown in Figure 6.7.

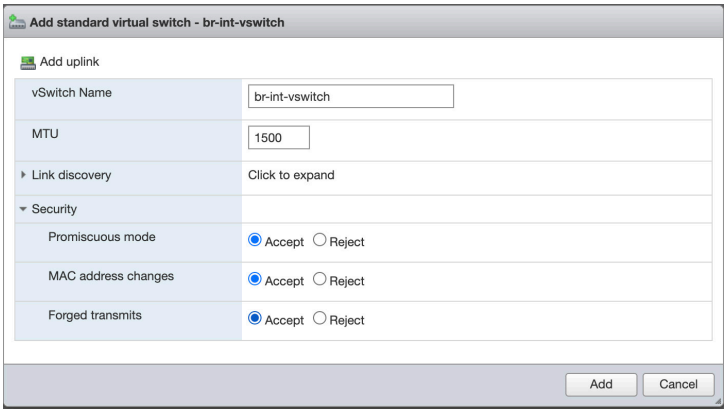

Figure 6.7 Internal Network Step 3

4. Create a port-group for br-int by selecting port groups under Networking tab and click on Add port group as shown in Figure 6.8.

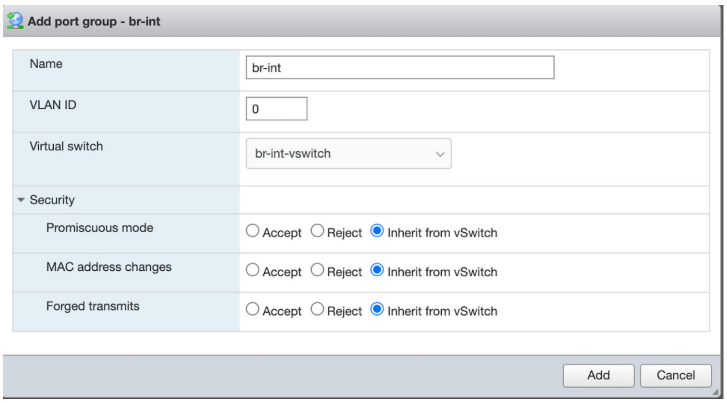

Figure 6.8 *Internal Network Step 4*

Ensure that the security options are inherited from the vSwitch. Click on Add and you now have a port group called *br-int-vmx1* with no adapters assigned.

Data Network

Now let's add a data network. Repeat this procedure as per the number of data NICs that you wish to add.

Let's create a single adapter named *WAN*.

1. Go to Networking and select Virtual switches tab as shown in Figure 6.9. Select Add standard virtual switch.

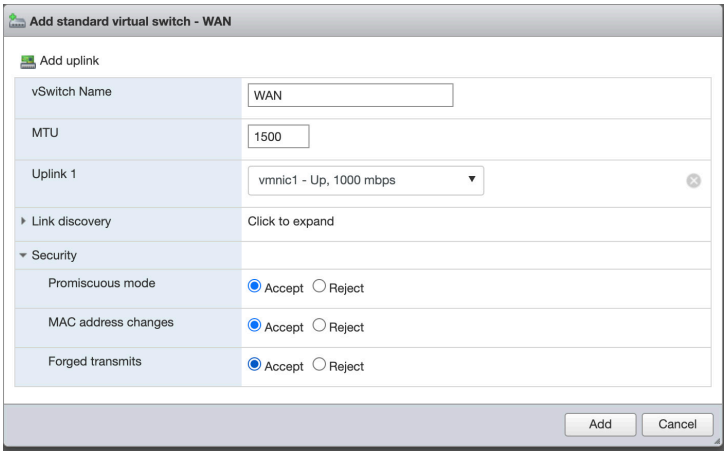

Figure 6.9 *Data Network Step 1*

Create a virtual switch with a name as WAN and add a physical NIC that can be used for data network.

2. Click Add to create the virtual switch. You can verify that the newly created virtual switch WAN is displayed.

Figure 6.10 *Data Network Step 2*

3. Create a port group for WAN by selecting port groups under Networking tab and click on Add port group.

4. Enter the name for the port group and ensure all security options are selected as shown in Figure 6.11. Click Add.

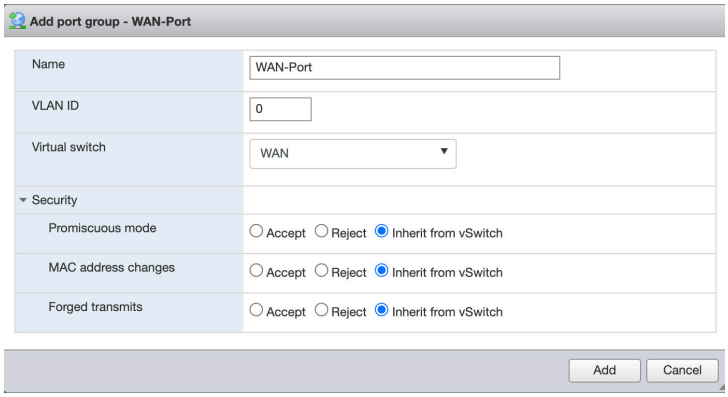

Figure 6.11 *Data Network Step 3 and 4*

You can see that the data network port is created in Figure 6.12.

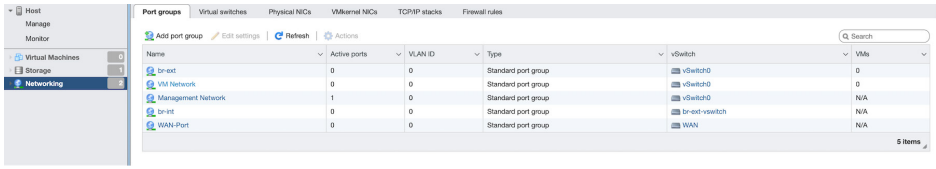

Figure 6.12 *Data Network Step 4*

NOTE To join two vMXs together on the same VMware system, use the following steps:

1. Repeat the Internal Network steps to create another vSwitch. You do not need to add any physical NICs to the vSwitch (this is the same configuration as the Internal vSwitch).
2. Add the VFP data interface for each vMX to this vSwitch. The process to set up the VFP interfaces is below.

You can now see that the following three networks are displayed in the networking summary screen: br-ext, br-int, and WAN.

NOTE You must enable promiscuous mode in all vSwitches so that packets with any MAC addresses can reach the vMX. This configuration is needed for OSPF to work properly.

Set Up the vMX Virtual Machines

Just like vMX on KVM, there are two VMs that must be created – the VCP running the Junos OS, and the VFP running an x86 virtualized release of Trio running on Wind River Linux.

The process for creating both of the VMs is very similar. It's a simple case of following the VMware wizard and choosing the correct settings for the VM.

Let's get started with the VCP.

VCP VM

Here are the steps required to create the VCP virtual machine (VM).

1. Select the Virtual Machines and select the **Create/Register VM** option in Figure 6.13.

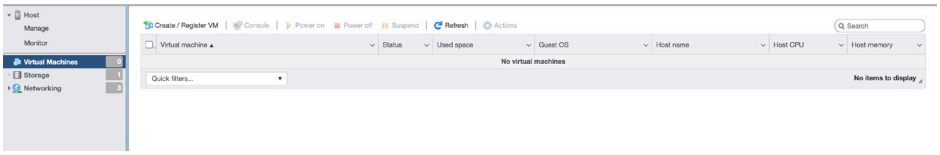

Figure 6.13 Select Virtual Machine Step 1

2. Select Deploy Virtual Machine from an OVF or OVA option shown in Figure 6.14 and click Next to procced.

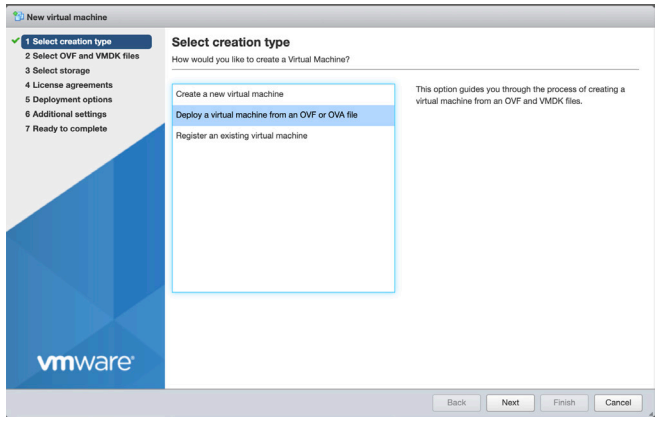

Figure 6.14 *Deploy Virtual Machine from an OVF or OVA Option Step 2*

3. Enter the name (example: vcp-vmx1) for the VM and upload the OVA file as shown in Figure 6.15.

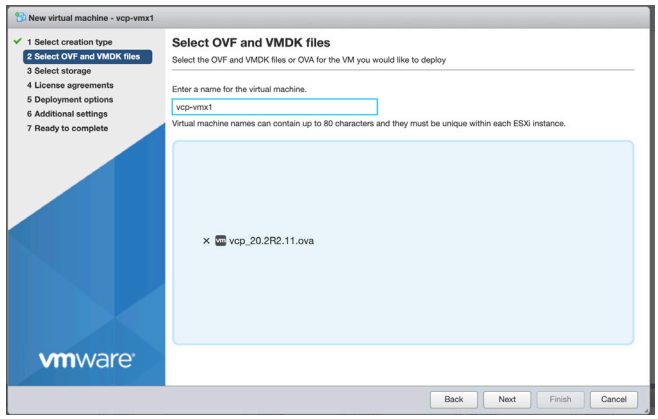

Figure 6.15 *Upload OVA File Step 3*

4. Select the storage and click Next as in Figure 6.16.

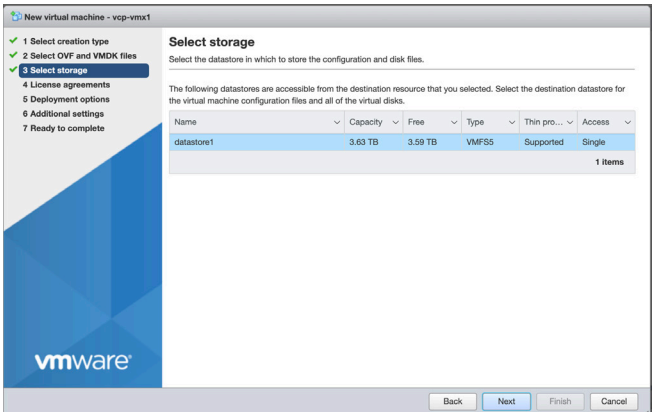

Figure 6.16 *Select Storage Step 4*

5. In the Deployment options pane (Figure 6.17), Network Mapping allows you to map the destination network on the host to the source network.

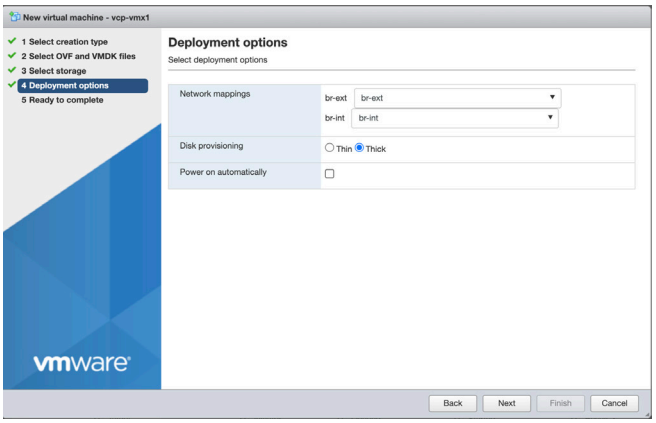

Figure 6.17 *Select Network Mapping Step 5*

For VFP, map the source networks as following: br-ext:

- external management network for connecting to management interfaces
- br-int: internal connection between VCP and VFP

6. After you select the destination network br-ext and br-int, select Thick Provisioning and click Next. In the Ready to complete pane (Figure 6.18), Verify the Configuration for the instance.

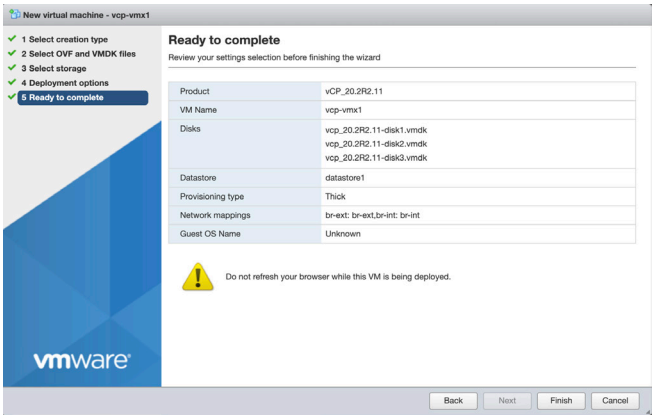

Figure 6.18 *Verify Configuration Step 6*

7. Click Finish (Figure 6.19) to complete creating the VCP (VM).

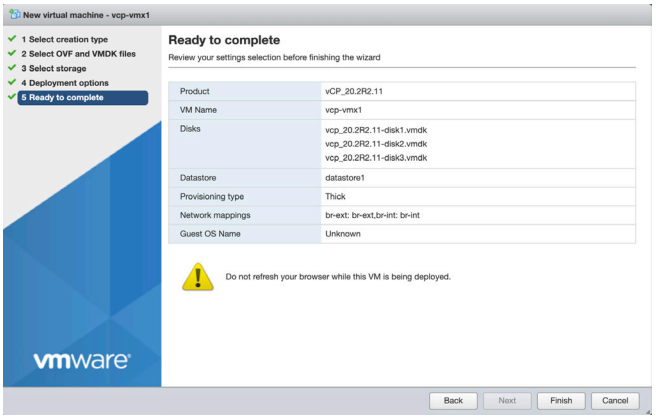

Figure 6.19 *Complete VCP Creation Step 7*

The .ova file is deployed as the VCP VM.

VFP VM

The process for creating the VFP is similar to the VCP. The process here outlines the steps required to create the VFP VM.

1. Select the Virtual Machines and select Create/Register VM option (Figure 6.20).

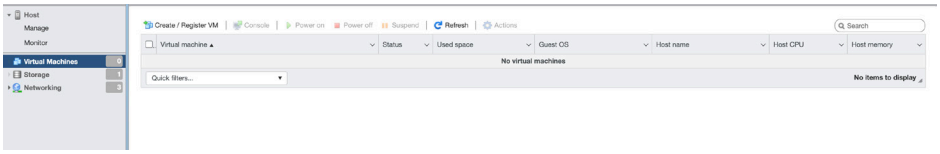

Figure 6.20 Select Virtual Machine Step 1

2. Select Deploy Virtual Machine (Figure 6.21) from an OVF or OVA and click Next to procced.

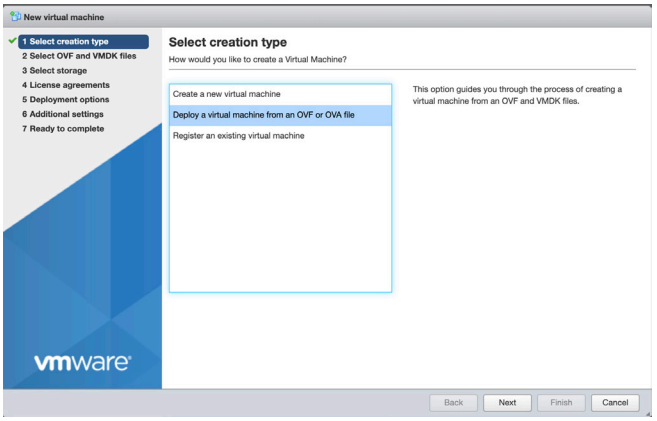

Figure 6.21 Deploy Virtual Machine from an OVF or OVA Option Step 2

3. Enter the name for the VM and upload (Figure 6.22) the OVA File. Let's use the name vfp-vmx1 in this example.

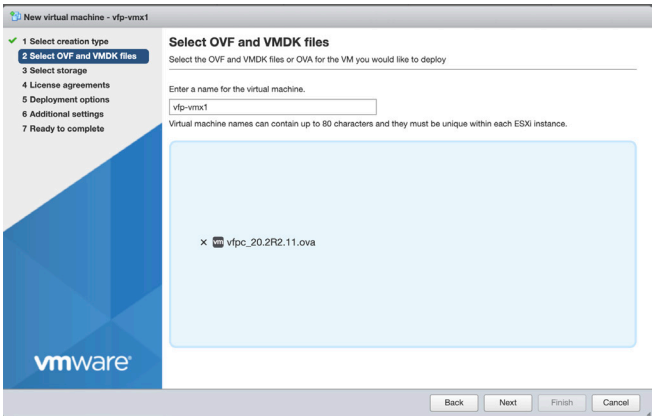

Figure 6.22 Upload OVA File Step 3

4. Select the storage (Figure 6.23) and click Next.

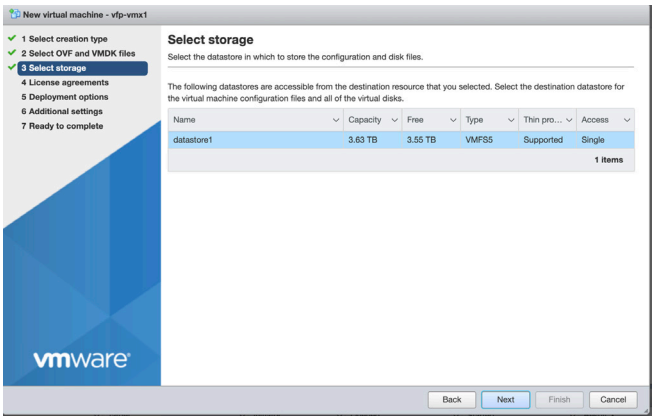

Figure 6.23 Select Storage Step 4

5. In the Deployment options (Figure 6.24), the Network mapping pane allows you to map the destination network on the host to the source network.

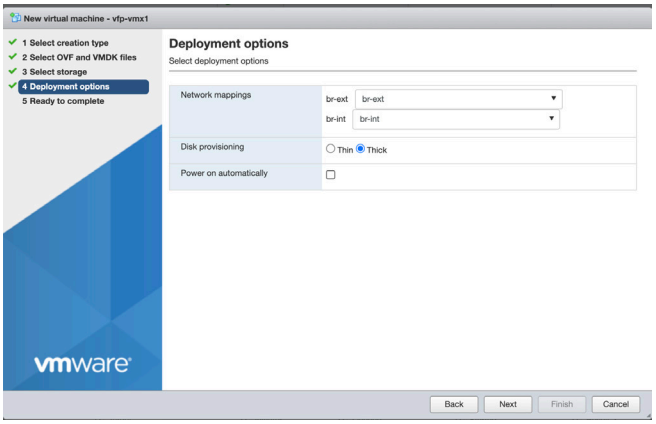

Figure 6.24 Select Network Mapping Step 5

For VFP, you must map the source networks for:

- br-ext: External management network for connecting to management interfaces
- br-int: Internal connection between VCP and VFP

After you select the destination network br-ext and br-int, select Thick Provisioning and click Next.

6. In the Ready to Complete pane (Figure 6.25), verify your configuration and click Finish.

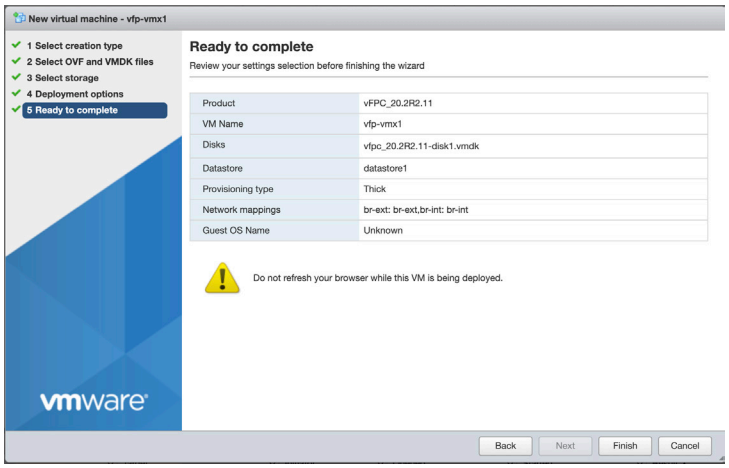

Figure 6.25 *Verify Configuration Step 6*

The .ova file is deployed as the VFP VM.

7. After you have deployed the VFP VM, you can modify the amount of memory, the number of vCPUs, and the number of WAN ports by selecting VFP instance and navigate to Actions tab and select Edit settings option (Figure 6.26).

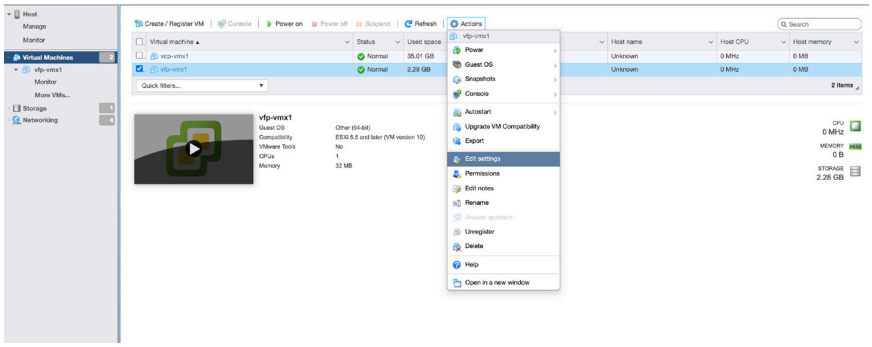

Figure 6.26 *Settings for VFP Step 7*

8. Before you launch the VFP VM, make sure that you have configured the proper number of vCPUs and memory for your VM based on the requirements described in minimum hardware and software requirements.

Change the CPU value as required (Figure 6.27).

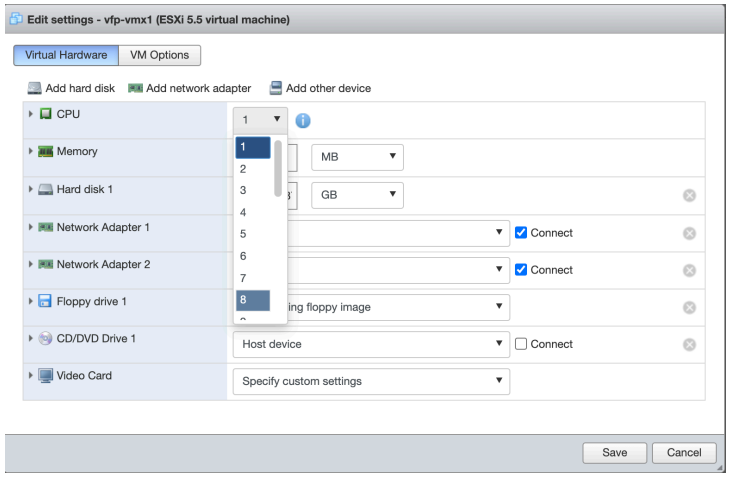

Figure 6.27 Change CPU Value Step 8

9. Change the memory as required (Figure 6.28).

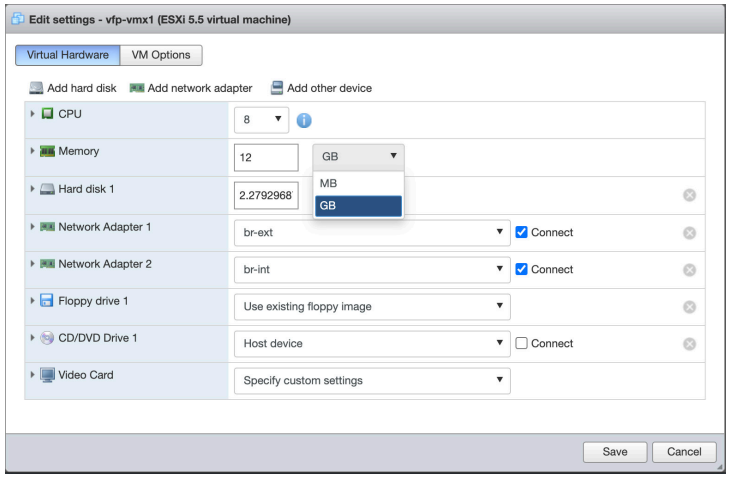

Figure 6.28 Change Memory Value Step 9

10. To add WAN ports, click Add network adapter in the Edit Settings (Figure 6.29) and Map it to respective Port Group.

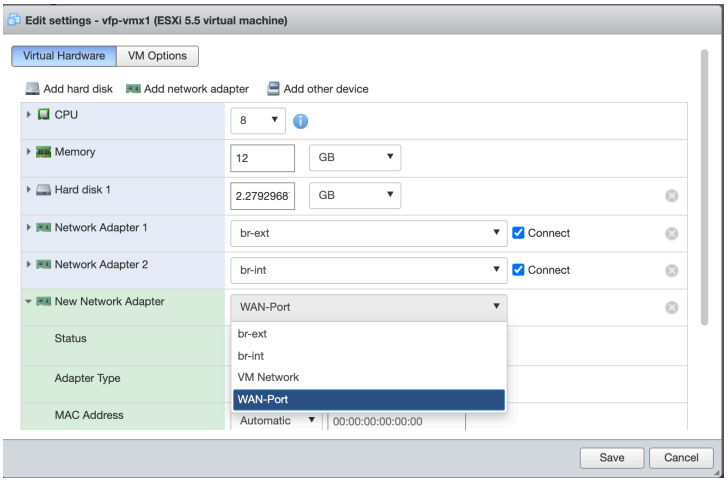

Figure 6.29 Add WAN Ports Step 10

11. For Adapter Type, select VMXNET3 (Figure 6.30) and click Next.

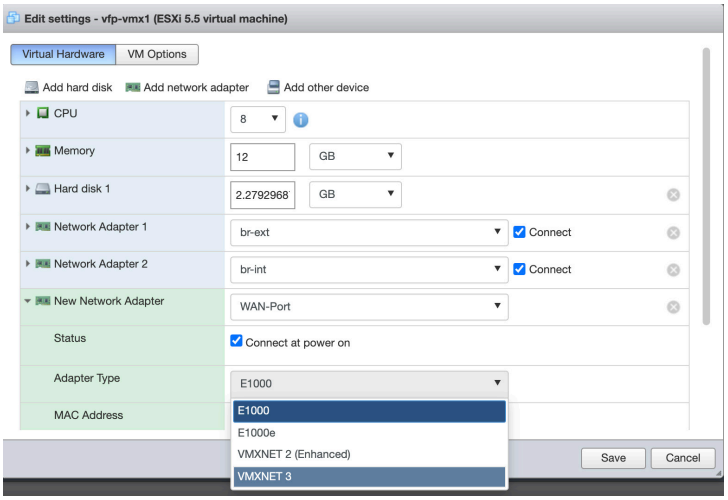

Figure 6.30 Select Adapter Type Step 11

12. Verify your configuration and click Save.

Serial Console

To aid with the troubleshooting and configuration of the vMX on VMware you should now set up a serial port connection to each VM so you can connect to the serial console of the VCP and VFP. This is accomplished by redirecting a telnet session to the serial port on the VM and is configured on VMware with the following steps.

1. Your vMX VMs will need to be stopped before you can complete all of these steps, so if you have not already done so then stop both VMs now.
2. In the Vsphere Web UI, select the networking option and click on firewalls rules on the right panel (Figure 6.31).

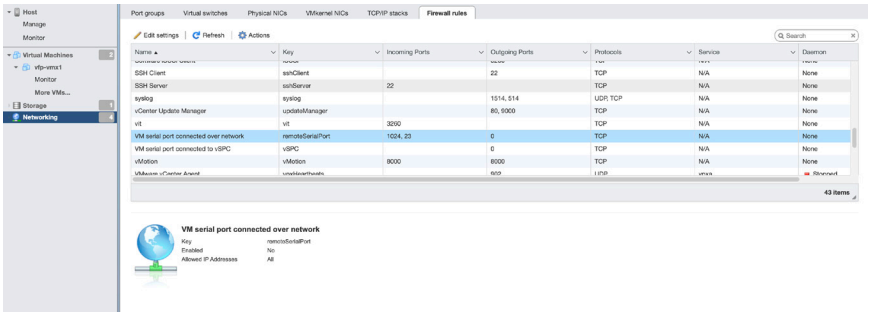

Figure 6.31 Select Networking Options Step 2

3. Select VM serial port connected over network and click on Edit Settings. The following (Figure 6.32) pop-up window is displayed.

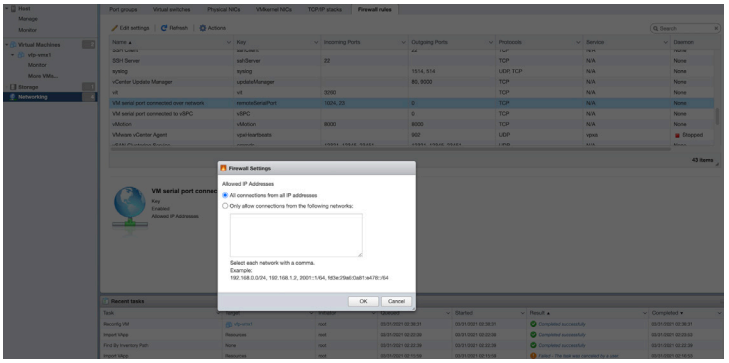

Figure 6.32 Settings for VM Serial Port Step 3

Select All connections from all IP addresses and click OK.

4. Now you can add the serial port to each vMX VM. In the left pane, select the VCP VM and click on Actions and click Edit settings (Figure 6.33).

5. Click Add other device on Edit settings and select Serial port.

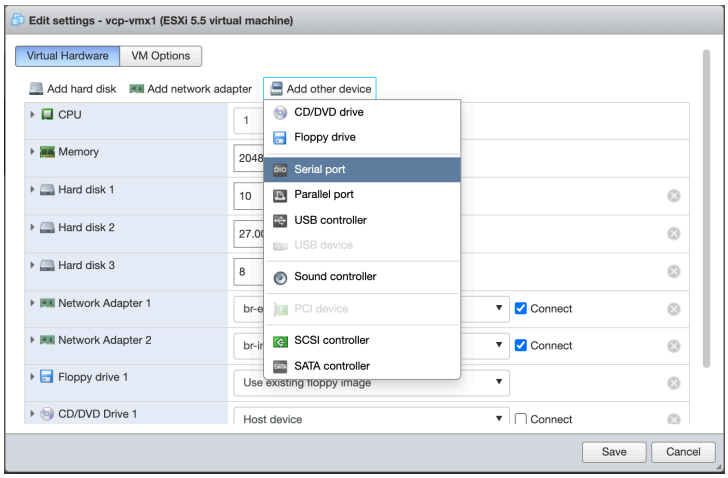

Figure 6.33 Add Other Devices for VM Serial Port Step 5

6. Under New Serial Port, select Direction: Server; and Port URL: telnet://:port-number.

7. Example: To use port 8601 for the serial connection on the VCP, type telnet://:8601 into the Port URL box (Figure 6.34).

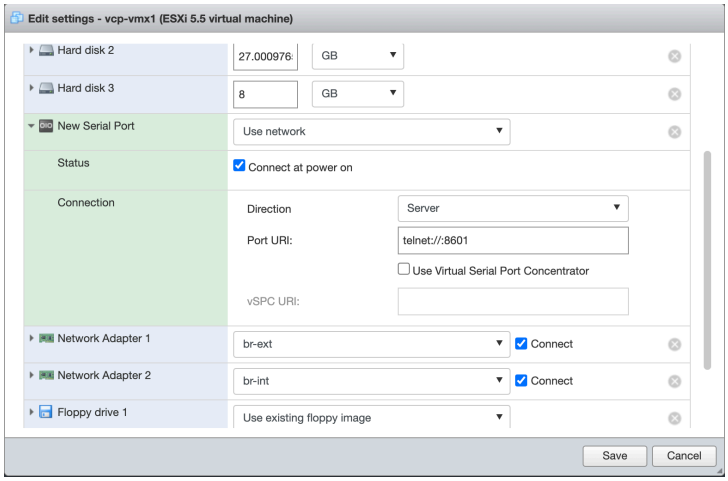

Figure 6.34 Edit Settings for New Serial Port Step 5

Ensure that Connect at power on is selected and click Next.

8. Repeat steps 5 through 7 for the VFP VM, this time choosing a different port number in step 7 (Figure 6.35).

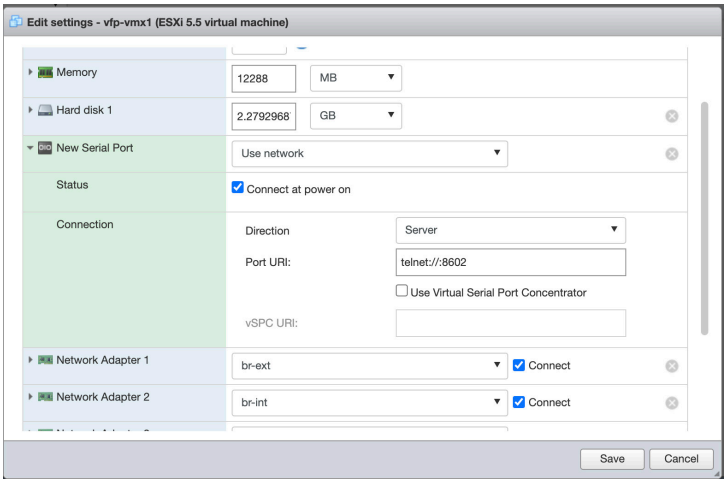

Figure 6.35 *Add Port Number for New VFP VM*

9. You can now use telnet to access the VCP or VFP serial ports by connecting to the telnet port specified in step 7 above.

NOTE Be aware that your VMware license may not permit you to use remote serial ports.

Verification

At this point if both machines have powered on successfully, you should have a running vMX.

Power on both the Instances (Figure 6.36).

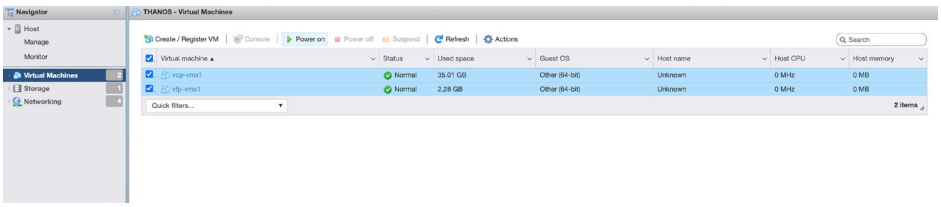

Figure 6.36 *Powering on VMs*

Open console tab for both the instances (Figure 6.37).

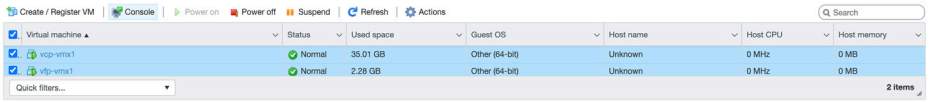

Figure 6.37 Open Console Tabs VMs

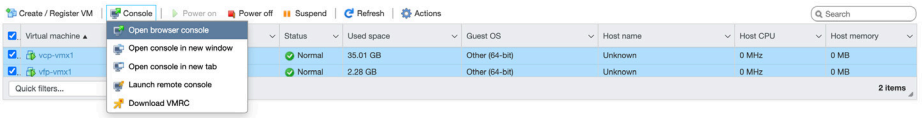

Figure 6.38 Console Tabs VMs

After you load the VCP VM console, log in, and run the Junos OS commands.

Installation of SR-IOV

If you have a physical NIC that supports SR-IOV, you can attach SR-IOV-enabled vNICs or virtual functions (VFs) to the vMX instance to improve performance. Before you install SR-IOV, ensure that the required NIC has SRIOV or PCI Passthrough enabled. You can verify the status from the WEB UI or vSphere Client.

To enable SR-IOV on a physical adapter follow these steps.

1. In vSphere Client, navigate to the host in the left navigation pane and click the Manage tab. Select Hardware tab and select PCI Devices (Figure 6.39). Select the physical adapter and click Configure SR-IOV option.

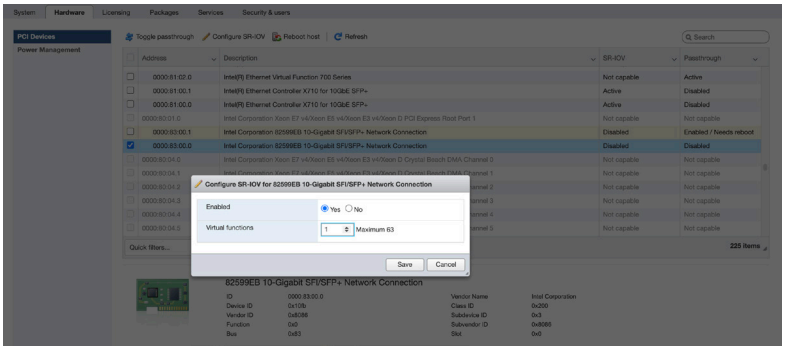

Figure 6.39 Select Configure SR-IOV Option Step 1

2. In the populated window, Select Yes to enable SR-IOV. In the number of virtual functions text box, specify the number of virtual functions to configure for the adapter (figure 6.40).
3. Click Save.
4. Restart the system after you enable the SR-IOV for the selected NIC.

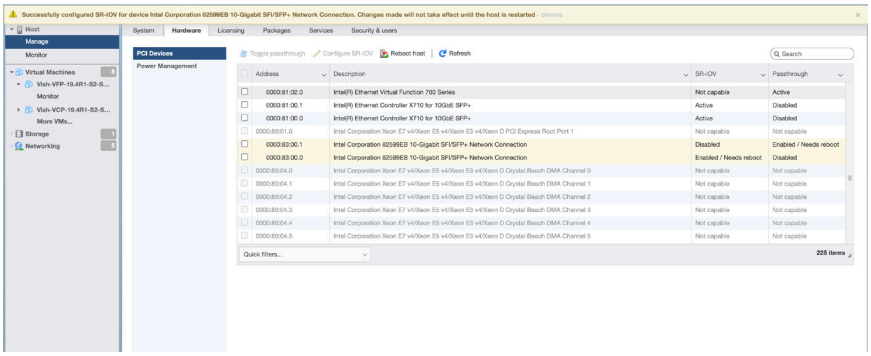

Figure 6.40 Restart After Enabling SR-IOV Step 4

5. Post reboot, you can verify that the status of SR-IOV is displayed as Active (Figure 6.41).

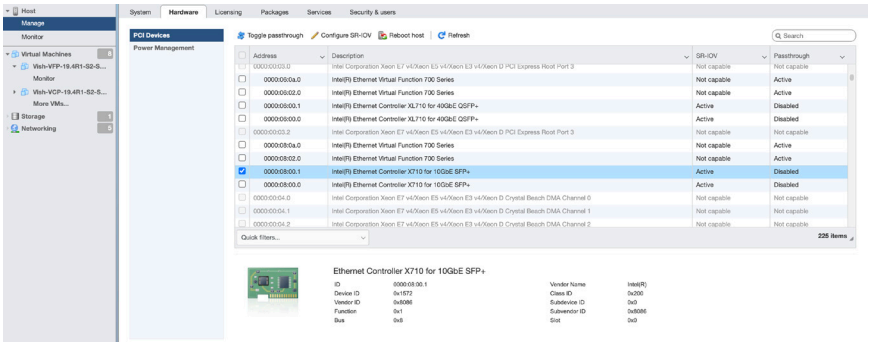

Figure 6.41 Verify SR-IOV Status Step 5

Now the SR-IOV for the selected NIC is ready.

Adding SR-IOV Interface to VFP VM

Create a vSwitch and port group for SR-IOV interface in vSphere Web UI:

Create a virtual switch by navigating to Networking option, select Virtual Switches and click Add standard virtual switch option (Figure 6.42).

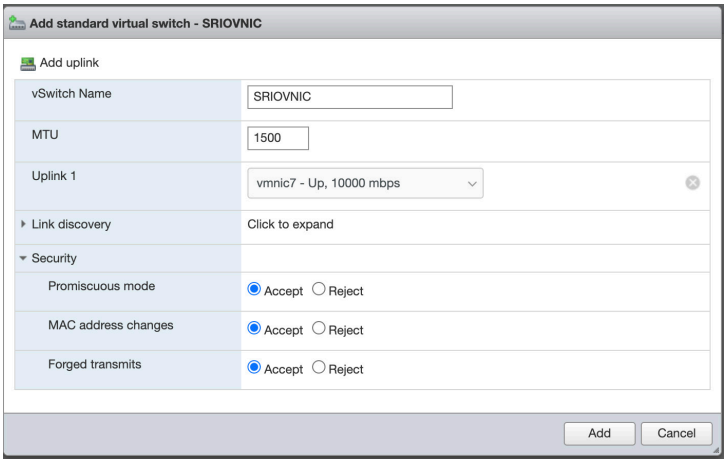

Figure 6.42 Add Standard Virtual Switch

Create a port group by navigating to Networking and select Port groups option and click Add port group (Figure 6.43).

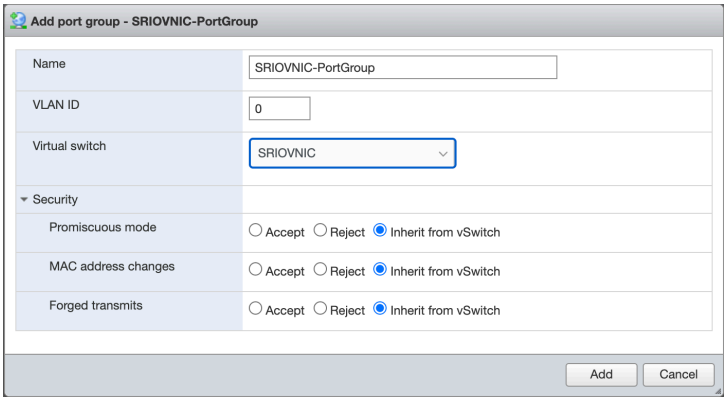

Figure 6.43 Add Port Group

To assign the SR-IOV to the VFP VM in vSphere Web UI:

1. Power-off the VFP VM while adding SR-IOV to the VFP VM.
2. Add SR-IOV ports by clicking Add network adapter option in the Edit settings page and map it to respective port group (see Figure 6.44).

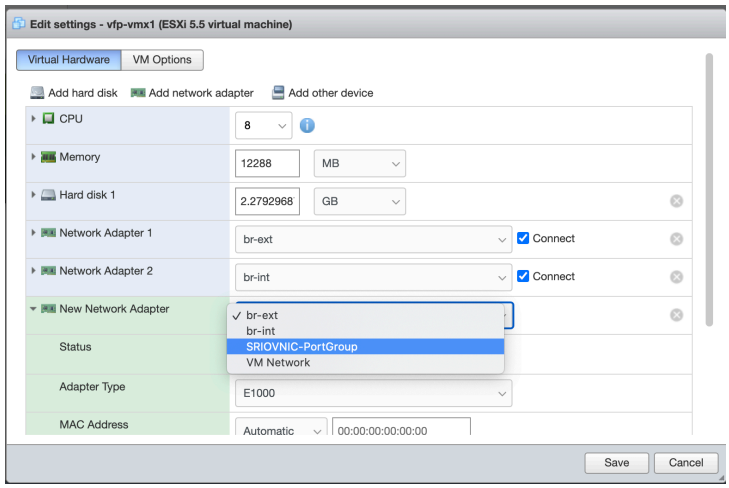

Figure 6.44 Add SR-IOV Ports

Change the Adapter Type to SR-IOV passthrough (Figure 6.45).

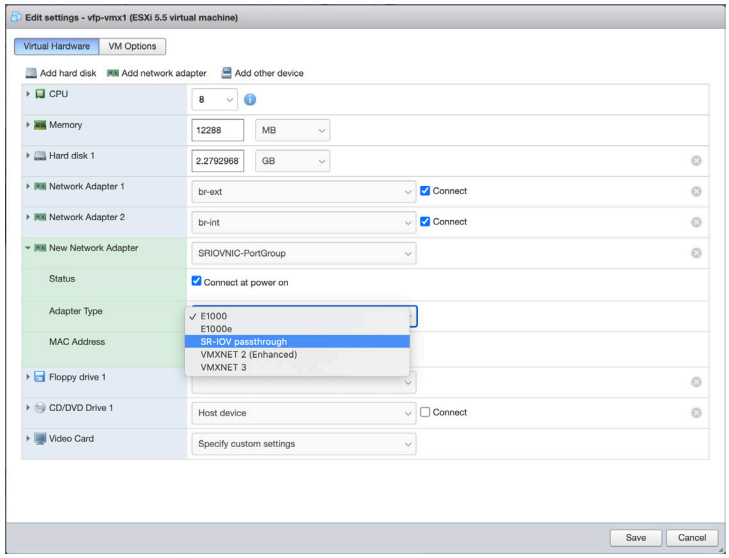

Figure 6.45 Select Adapter Type

Map the Physical function to respective the SR-IOV NIC and save changes (Figure 6.46).

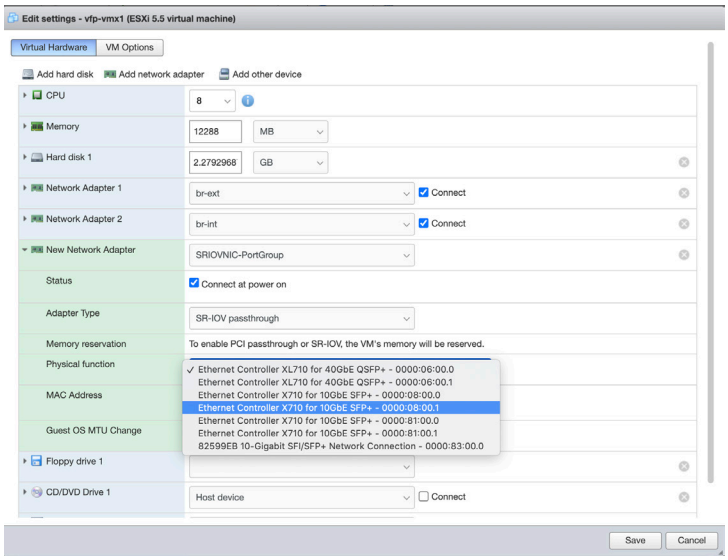

Figure 6.46 Map Physical Function

Now the SR-IOV interface is added to the VFP VM.

Chapter 7

vMX Modified and Unmodified Drivers

This chapter explains the physical and virtual component communication in vMX, namely virtio and SR-IOV, but also highlights the differences in modified and unmodified drivers of a Network Interface Card (NIC). The NIC driver is software that translates the language of the NIC to the operating system commands and it's worth knowing whether you should use the driver software as is or use the Juniper-provided driver software for your vMX for the best deployment.

Virtio and SR-IOV Usage

You can enable communication between a Linux-based virtualized device and VM instances either by using virtio or by using suitable hardware and single-root I/O virtualization (SR-IOV).

Virtio is part of the standard libvirt library of helpful virtualization functions and is normally included in most versions of Linux. Virtio adopts a software-only approach. The virtio package supports block (storage) devices and network interface controllers.

Generally, using virtio is quick and easy. Libvirt is part of every Linux distribution and the commands to establish the bridges are well-understood. However, virtio places all of the burden of performance on the host OS, which normally bridges all the traffic between virtual network functions (VNFs), into and out of the device.

The overall architecture of virtio is illustrated in Figure 7.1

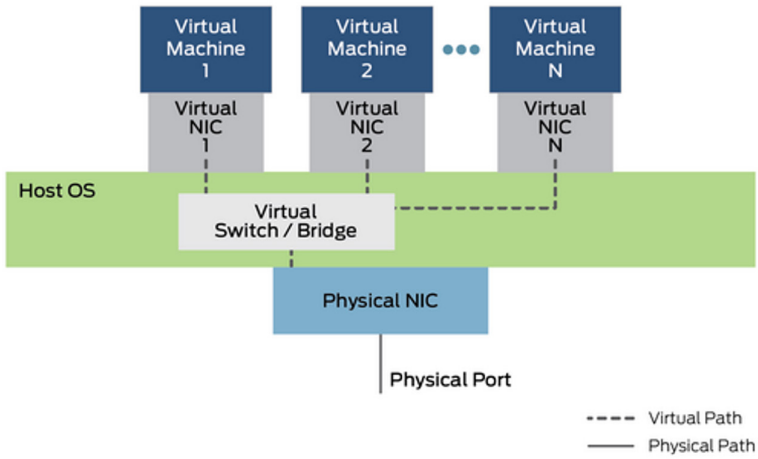

Figure 7.1 Virtio Architecture

Single root I/O virtualization (SR-IOV) allows a physical function to appear as multiple, separate vNICs. SR-IOV allows a device, such as a network adapter, to have separate access to its resources among various hardware functions. If you have a physical NIC that supports SR-IOV, you can attach SR-IOV-enabled vNICs or virtual functions (VFs) to the vMX instance to improve performance.

The architecture of SR-IOV is shown in Figure 7.2.

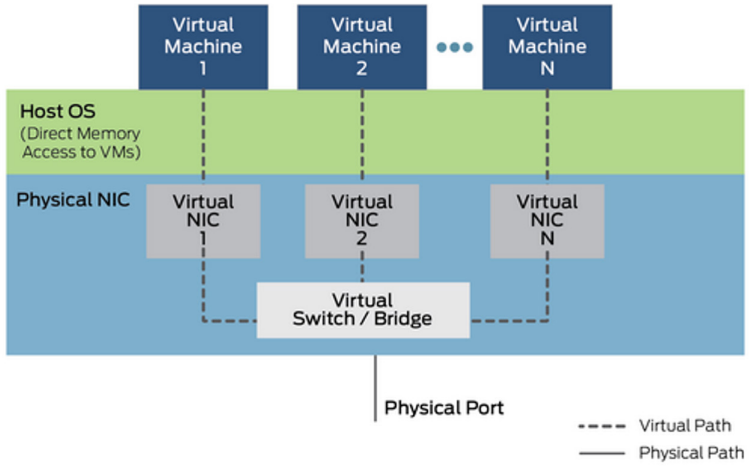

Figure 7.2 SR-IOV Architecture

SR-IOV requires software written in a certain way and makes use of specialized hardware, which means an increase in cost, even with a simple device. SR-IOV on the vMX for KVM requires one of the following Intel NIC drivers:

- Intel X520 or X540 using 10 G ports and ixgbe driver
- Intel X710 or XL710 using 10 G ports and i40e driver
- Intel XL710Q-DA2 NIC with 40 G ports using i40e driver

Modified and Unmodified Drivers

What's the difference between modified drivers and unmodified drivers?

- *Unmodified driver*: The ixgbe/i40e device driver software is available for download from the Intel website and you can install it on vMX.
- *Modified driver*: The ixgbe/i40e device driver software comes bundled with vMX software. Juniper Networks provides the driver software to enable MAC promiscuous and VLAN promiscuous mode to support Layer 2 forwarding functionality.

Starting in Junos OS Release 18.4R1, you can deploy vMX instances with an unmodified i40e or IXGB drivers on Ubuntu version 16.04 or later.

Support for modified drivers for i40e is not available starting in Junos OS Release 19.1 and later releases.

Deploying vMX with Unmodified Driver

Let's take an example of deploying vMX with an Unmodified ixgbe driver. Before installing a vMX instance, you must choose to load the unmodified driver:

1. Download the software from Intel® Network Adapter Driver for PCIe* Intel® 10 Gigabit Ethernet Network Connections Under Linux and save it into any directory of your choice and follow the README instructions to proceed.
2. Remove the existing driver module:

```
sudo rmmod ixgbe
```

3. Install the driver software:

```
cd ~/intel_ixgbe/ixgbe-5.5.3/src
sudo  make clean
sudo make install
```

4. Install the required version of the unmodified driver on the host:

```
modprobe ixgbe
```

5. Use the ethtool -i interface-name utility to get the driver information:

```
[root@host ~]# ethtool –i eth6
driver: ixgbe
version: 5.3.6
firmware-version: 0x61bd0001
```

6. Create a virtual function (VF) using either of the following commands. Example. Create two VFs:

```
echo 2 > /sys/class/net/eth16/device/sriov_numvfs
```

7. Configure the vMX configuration file (vmx.conf) and ensure that use_native_drivers : true is set:

```
FORWARDING_PLANE:
    memory-mb   : 16384
    vcpus       : 12
    console_port: 8602
    device-type : sriov
    use_native_drivers : tru
```

8. Install vMX:

```
./vmx.sh --install --cfg ../vmx.conf
```

The vMX programs the PF driver with VLAN information. The PF driver compares the outer VLAN of the VLAN tag information of the packets against the programmed VLAN and forwards it to the corresponding VF.

9. Enter the CLI configuration mode after logging in to the vMX and set the per interface configuration knob for the respective interface.

```
set interfaces <interface-name> vlan-offload
```

Changing from Unmodified i40e Driver to Modified i40e Driver

When you try to move an existing deployment from unmodified IXGBE driver to modified IXGBE driver, perform the following steps:

1. Clear the relevant knob “use_native_drivers : true” from vMX configuration file:

```
FORWARDING_PLANE:
memory-mb   : 16384
vcpus       : 12
console_port: 8602
device-type : sriov
```

2. Clean up the vMX configuration:

```
./vmx.sh --cleanup --cfg ../vmx.conf
```

3. Reinstall vMX on your device:

```
./vmx.sh --install --cfg ../vmx.conf
```

MORE? To understand details about the supported features for modified and unmodified drivers, see the KVM topic in vMX Getting Started Guide: https://www.juniper.net/documentation/us/en/software/vmx/vmx-getting-started/topics/concept/features-supported-modified-unmodified-drivers.html.

This information can also help you decide which driver you might want to use for your vMX deployment.

Book End Summary

Now that you've learned how to build and configure vMX, why not go ahead and deploy the vMX router to meet your own specific requirements? The vMX supports the DCI and Layer 2/Layer 3 technologies that are available on the physical MX and if a feature becomes available on MX, it will push down to the vMX. Perhaps the vMX router will enable you to quickly introduce a new service or sandbox test a new configuration.

Here are some examples of use cases for vMX:

- Service Provider Edge – a virtual MPLS PE in scale out deployment scenarios, or as a peering router.
- Data Center Gateway – a gateway router that is capable of supporting the different DC overlay, DC interconnect, and L2 technologies used in the DC such as GRE, VXLAN, VPLS, and EVPN.
- Enterprise WAN router – an Internet gateway or for proving an overlay network over a service providers MPLS or Layer 2 network.
- Virtual Route Reflector.
- Virtual Broadband Network Gateway (vBNG) within Service Provider infrastructure.
- Simulation and configuration testing in your lab environment.

MORE? All things vMX are here on Juniper.net: https://www.juniper.net/us/en/products-services/routing/mx-series/vmx/

Let's now review what you've accomplished by reading this *Day One* book. You built an Ubuntu KVM host and then configured an instance of vMX in a simple lab topology. This topology was then scaled up from a simple four router topology to a topology consisting of eight routers that could be easily scaled to thirty routers and beyond. Some cool features of Junos were configured – such as VPLS and EVPN.

At this point why not try to scale the topology further – go ahead and add more routers, and perhaps learn about a different protocol – you could take the final topology and remove OSPF, but using IS-IS as the IGP instead?

You now have a working MPLS installation, so why not get more familiar with it? Add a few more P routers, and maybe play with Traffic Engineering LSPs and force traffic over a particular path. But most importantly get familiar with troubleshooting Junos on the lab topology.

The topology you built can now be extended to support even the most complicated JNCIE configuration, so it's time to go ahead with vMX on your own. Have fun deploying vMX in your own environment!

Appendix

Ten Most Active vMX Support Issues

1. Interface ordering workaround for vMX on ESXi server:

https://kb.juniper.net/InfoCenter/index?page=content&id=KB36433&cat=MX_SERIES&actp=LIST

2. Connect SR-IOV interface to physical CPE for vMX in performance-mode:

https://kb.juniper.net/InfoCenter/index?page=content&id=KB36486&cat=MX_SERIES&actp=LIST

3. Checking vMX deployed with SR-IOV interface on an RHOSP cluster:

https://kb.juniper.net/InfoCenter/index?page=content&id=KB36505&cat=MX_SERIES&actp=LIST

4. The vMX is not accepting the correct license keys (release 19.4R2):

https://kb.juniper.net/InfoCenter/index?page=content&id=KB36478&cat=VMX&actp=LIST

5. Enable trust mode for SR-IOV ports on the host of vMX instances:

https://kb.juniper.net/InfoCenter/index?page=content&id=KB36100&cat=MX_SERIES&actp=LIST

6. Check the SR-IOV capability of NICs in vMX devices:

https://kb.juniper.net/InfoCenter/index?page=content&id=KB34884

7. Collect the VFP logs and copy them to VCP in vMX:

https://kb.juniper.net/InfoCenter/index?page=content&id=KB34887

8. How to log in to a vFPC instance on the vMX instance using SSH:

https://kb.juniper.net/InfoCenter/index?page=content&id=KB34874

9. Enabling hardware-assisted virtualization in ESXi by using vSphere client:

https://kb.juniper.net/InfoCenter/index?page=content&id=KB32951

10. vMX on ESXi issues in performance mode with latest processors (Intel Ivy Bridge processors or later):

https://kb.juniper.net/InfoCenter/index?page=content&id=KB32959