Virtual Route Reflector

vRR Getting Started Guide for KVM

Published
2021-10-05

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at https://support.juniper.net/support/eula/. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

About This Guide

Use this guide to install the virtual Route Reflector in the KVM environment. This guide also includes basic vRR configuration and management procedures.

After completing the installation and basic configuration procedures covered in this guide, refer to the Junos OS documentation for information about further software configuration.

RELATED DOCUMENTATION

Using Route Reflectors for BGP Networks

1

CHAPTER

Virtual Route Reflector Overview

Understanding Virtual Route Reflector

IN THIS SECTION

The virtual Route Reflector (vRR) feature allows you to implement route reflector capability using a general purpose virtual machine that can be run on a 64-bit Intel-based blade server or appliance. Because a route reflector works in the control plane, it can run in a virtualized environment. A virtual route reflector on an Intel-based blade server or appliance works the same as a route reflector on a router, providing a scalable alternative to full mesh internal BGP peering. The vRR feature has the following benefits:

- Scalability: By implementing the vRR feature, you gain scalability improvements, depending on the server core hardware on which the feature runs. Also, you can implement virtual route reflectors at multiple locations in the network, which helps scale the BGP network with lower cost.
- Faster and more flexible deployment: You install the vRR feature on an Intel server, using open source tools, which reduces your router maintenance.
- Space savings: Hardware-based route reflectors require central office space. You can deploy the vRR feature on any server that is available in the server infrastructure or in the data centers, which saves space.

Virtual Route Reflector Package Contents

The vRR software packages are available as these types of packages:

- Application package—This package is for launching vRR software in a virtualized environment for the first time.
- Install package—This package is for upgrading vRR software that is already running to the next Junos OS release.

  Starting with Junos OS Release 15.1, the install package for vRR (**jinstall64-vrr-*.***) is no longer available. Use the install package of Junos OS for MX Series platforms: **junos-install-mx-x86-64-*.tgz** (e.g. use 64 Bit-MX High-End Series for MX240: Downloads for MX240) to upgrade vRR.

The vRR software images are available in these flavors:

- KVM and OpenStack—TGZ package
- VMware ESXi—OVA package
- Unified—64-bit Junos OS (upgraded FreeBSD kernel)
- Legacy—64-bit Junos OS

  Starting with Junos OS Release 15.1, the legacy package (**jinstall64-vrr-*.***) is no longer available.

Virtual Route Reflector Restrictions

The following features are not supported with the vRR feature:

- Graceful Routing Engine Switchover (GRES)
- Nonstop Active Routing (NSR)
- Unified in-service software upgrade (unified ISSU)

vRR is qualified primarily as a route reflector with minimal data plane support. For packet forwarding, MPLS VPN, and CoS feature support, you might consider vMX.

Release History Table

Release	Description
16.1	Starting with Junos OS Release 16.1, use the KVM archive (vrr-bundle-kvm-*.tgz) for vRR deployments on Linux hosts.
15.1	Starting with Junos OS Release 15.1, the install or the legacy package for vRR (jinstall64-vrr-*.*) is no longer available. For Junos Releases 15.1 >= Junos Os Releases < 16.1, use the unified package (junos-x86-64-*.vmdk).

Virtual Route Reflector Hardware Requirements

Table 1 on page 4 lists the hardware requirements.

Table 1: Hardware Requirements

Description	Value
CPU	Intel Xeon Nehalem or newer generation processor
Memory	8 GB for vRR to run with default settings 32 GB for vRR to achieve higher scale
Storage	Local or NAS Each vRR instance requires 25G of disk storage
Other requirements	Hyperthreading (recommended) Any ESXi HCL supported NIC

Virtual Route Reflector Software Requirements

You can install vRR on Linux systems using KVM and libvirt.

Table 2 on page 5 lists the supported operating systems for the host.

Table 2: Supported Operating Systems

Operating System	Releases
Ubuntu	Starting with vRR Release 19.1R1, Ubuntu 16.04 with QEMU-KVM 1:2.5+dfsg-5ubuntu10.43 and libvirt 1.3.1 is supported. For vRR releases before 19.1R1, Ubuntu 14.04 with QEMU-KVM 2.0.0+dfsg-2ubuntu1.11 and libvirt 1.2.2 is supported. • Ubuntu 14.04 QEMU-KVM 2.0.0+dfsg-2ubuntu1.11 libvirt 1.2.2 • Ubuntu 16.04 QEMU-KVM 1:2.5+dfsg-5ubuntu10.43 libvirt 1.3.1
CentOS	• CentOS 7.1 QEMU-KVM 1.5.3 libvirt 1.2.8 • CentOS 7.2 QEMU-KVM 1.5.3 libvirt 1.2.17
Red Hat Enterprse Linux	• Red Hat Enterprise Linux 7.1 QEMU-KVM 1.5.3 libvirt 1.2.8 • Red Hat Enterprise Linux 7.2 QEMU-KVM 1.5.3 libvirt 1.2.17

To install vRR, you must also install these packages:

- virt-manager
- bridge-utils

Release History Table

Release	Description
19.1R1	Starting with vRR Release 19.1R1, Ubuntu 16.04 with QEMU-KVM 1:2.5+dfsg-5ubuntu10.43 and libvirt 1.3.1 is supported. For vRR releases before 19.1R1, Ubuntu 14.04 with QEMU-KVM 2.0.0+dfsg-2ubuntu1.11 and libvirt 1.2.2 is supported.

CHAPTER 2

Installing and Configuring Virtual Route Reflector on KVM

Installing the Virtual Route Reflector Image on KVM

IN THIS SECTION

Before you install vRR:

1. Download the vRR software package (**vrr-*.tgz**) from the Virtual Route Reflector page and uncompress the package in a location accessible to the server.

2. (For Ubuntu) Prepare the Ubuntu host by disabling APIC virtualization.

   Edit the **/etc/modprobe.d/qemu-system-x86.conf** file and add `enable_apicv=0` to the line containing `options kvm_intel`.

   ```
   options kvm_intel nested=1 enable_apicv=0
   ```

   Reboot the host or unload and reload the kernel module.

3. (For CentOS) Copy the vRR image to the **libvirtd** directory and rename it with the name of your VM.

   ```
   cp vrr-image-name vrr-vm-filename
   ```

   NOTE: For Junos Releases 15.1 >= Junos Os Releases < 16.1, use the unified package. Convert the **vmdk** image to **qcow2** format using the `qemu-img convert -f vmdk -O qcow2` *`vmdk-filename qcow2-filename`* command. For example: **qemu-img convert -f vmdk -O qcow2 junos-x86-64-15.1R1.9.vmdk junos-x86-64-15.1R1.9.qcow2**

   Starting with Junos OS Release 16.1, use the VRR KVM bundle: **vrr-bundle-kvm-21.1R1.11.tgz**. Only the qcow2 image within the KVM bundle: **junos-x86-64-*.img** is required to bring up VRR (you don't have to use the metadata.img).

For example, these commands copy the download image to the **vrr-VM01.img** file in the **libvirtd/images** directory:

```
cp jinstall64-vrr-14.2R3.8-domestic.img /var/lib/libvirtd/images/vrr-VM01.img
cp junos-x86-64-17.3R1.10.qcow2 /var/lib/libvirtd/images/vrr-VM01.img
```

To install vRR, perform these tasks:

Configuring the Linux Bridges

You must set up these Linux bridges for the vRR interfaces to have proper connectivity.

- em0 interface (for example, vrr-mgmt)
- em1 interface (for example, vrr-ext)

NOTE: The em0 interface can only function as a management interface. You cannot use the em0 interface for routing configurations.

For remote connectivity to the vRR instance, you can add physical interfaces from the host.

The bridges are not persistent across reboots. To make them permanent, you must add them to the appropriate configuration files for your Linux distribution.

To configure the bridges:

1. Create the bridges.

   ```
   user@node:~$ brctl addbr vrr-mgmt
   user@node:~$ brctl addbr vrr-ext
   ```

   Verify that the bridges have been created with the `brctl show` command.

2. For each bridge, an interface with the same name is created on the system. Make sure these interfaces are in an Admin Up state.

   ```
   user@node:~$ ip link set dev vrr-mgmt up
   user@node:~$ ip link set dev vrr-ext up
   ```

Verify that the interfaces are up with the `ip link show` command.

3. To provide remote connectivity for the vRR instance, add physical interfaces to these bridges.

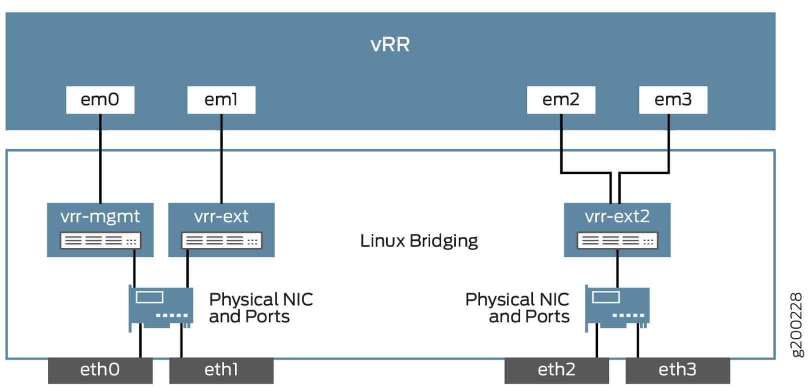
```
user@node:~$ brctl addif vrr-mgmt interface-name
user@node:~$ brctl addif vrr-ext interface-name
```

NOTE: KVM installations support virtio driver for the em1 and em2 interfaces.

Launching the vRR VM

The physical interfaces are mapped to the VM interfaces (such as em0) using Linux bridging. Figure 1 on page 10 illustrates this mapping. You can use an XML template or the `virt-install` utility to create this interface mapping when you launch the vRR VM.

Figure 1: vRR Interface Mapping



To launch the VRR instance:

1. You can use the `virsh create` command or the `virt-install` utility.

   - Use the `virsh create` *`vrr-instance-name`*`.xml` command with the XML template file.

     For example: `virsh create vrr.xml`

Here is a sample XML template file for **vrr.xml**.

```
<domain type='kvm' id='10'>
<!-- Assign VRR VM instance name -->
  <name>sample_vrr</name>
<!-- Assign Memory required for this VRR VM instance -->
  <memory unit='KiB'>25165824</memory>
  <currentMemory unit='KiB'>25165824</currentMemory>
  <memoryBacking>
    <locked/>
  </memoryBacking>
<!-- Assign required virtual CPU for VRR VM instance, here 4 vcpu is assigned -->
  <vcpu placement='static' cpuset='0-3'>4</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='1'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='3'/>
  </cputune>
  <resource>
    <partition>/machine</partition>
  </resource>
  <sysinfo type='smbios'>
    <bios>
      <entry name='vendor'>Juniper</entry>
    </bios>
    <system>
      <entry name='manufacturer'>Juniper</entry>
      <entry name='product'>VRR</entry>
      <entry name='version'>16.1</entry>
    </system>
  </sysinfo>
  <os>
    <type arch='x86_64' machine='pc-i440fx-1.7'>hvm</type>
    <boot dev='hd'/>
    <smbios mode='sysinfo'/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
<cpu mode='host-model'>
```

```
    <model fallback='allow'/>
    <topology sockets='1' cores='4' threads='1'/>
  </cpu>
  <clock offset='utc'/>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
<!-- Provide VRR VM instance image location -->
    <disk type='file' device='disk'>
          <driver name='qemu' type='qcow2' cache='writethrough'/>
          <source file='/var/tmp/junos-x86-64-21.1R1.11.img'/>
          <backingStore/>
          <target dev='vda' bus='virtio'/>
          <alias name='virtio-disk0'/>
          <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
      </disk>
    <controller type='pci' index='0' model='pci-root'>
      <alias name='pci.0'/>
    </controller>
<!-- em0 - management interface with the associated bridge -->
    <interface type='bridge'>
          <source bridge='vrr-mgmt'/>
      <target dev='vnet5'/>
      <model type='virtio'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
    </interface>
<!-- em1 - external data interface with the associated bridge -->
    <interface type='bridge'>
      <source bridge='vrr-ext'/>
      <target dev='vnet7'/>
      <model type='virtio'/>
      <alias name='net1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
    </interface>
<!-- em2 - external data interface with the associated bridge -->
    <interface type='bridge'>
      <source bridge='vrr-ext2'/>
      <target dev='vnet8'/>
      <model type='virtio'/>
      <alias name='net2'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
    </interface>
<!-- em3 - external data interface with the associated bridge -->
```

```
    <interface type='bridge'>
      <source bridge='vrr-ext2'/>
      <target dev='vnet9'/>
      <model type='virtio'/>
      <alias name='net3'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </interface>
    <serial type='pty'>
      <source path='/dev/pts/0'/>
      <target port='0'/>
      <alias name='serial0'/>
    </serial>
    <console type='pty' tty='/dev/pts/0'>
      <source path='/dev/pts/0'/>
      <target type='serial' port='0'/>
      <alias name='serial0'/>
    </console>
    <memballoon model='none'>
       <alias name='balloon0'/>
    </memballoon>
  </devices>
  <seclabel type='none' model='none'/>
</domain>
```

- (For Junos OS Release 14.1 or 14.2) You can use the `virt-install` utility.

```
virt-install --name vrr-instance-name \
--vcpus 1 \
--ram 8192 \
--import \
--disk path=./image-name,format=qcow2,bus=virtio \
--disk path=./metadata.img,format=qcow2,bus=virtio \
--serial tcp,host=0.0.0.0:5025,protocol=telnet \
--network bridge=vrr-mgmt,model=virtio \
--network bridge=vrr-ext,model=virtio
```

where:

--name	Specifies the name of the vRR instance.
--disk	Specifies the path to the image file.

For example:

```
virt-install --name vrr-142r1 \
--vcpus 1 \
--ram 8192 \
--import \
--disk path=./junos-x86-64-14.2R1.6.img,format=qcow2,bus=virtio \
--disk path=./metadata.img,format=qcow2,bus=virtio \
--serial tcp,host=0.0.0.0:5025,protocol=telnet \
--network bridge=vrr-mgmt,model=virtio \
--network bridge=vrr-ext,model=virtio
```

NOTE: After you have installed and started the vRR instance, you can access the serial console port for the VM using the Telnet protocol. For example: `telnet 127.0.0.1 5025`

2. You can connect to the VM console using the `virsh console` *`vrr-instance-name`* command.

   Wait for the system to boot and present the login prompt. You can log in and configure vRR as you would normally do on a router.

   To disconnect from the console, press Ctrl +].

3. Verify that your VM is installed as vRR using the `show version` command.

   NOTE: The model must appear as vrr.

   For example:

   ```
   root> show version
   ```

   ```
   Model: vrr
   ```

4. Verify that your VM is installed using the `show interfaces terse` command. The added interfaces appear as em interfaces. For example:

```
root> show interfaces terse
```

```
Interface               Admin Link Proto    Local                 Remote
...
em0                     up    up
em1                     up    up
...
```

RELATED DOCUMENTATION

Configuring Interfaces, Protocols, and Routes of the Virtual Route Reflector Using Junos CLI | 17

Launching a vRR VM Using the Unified Image

NOTE: For Junos Releases prior to 16.1, a unified image in generic **vmdk** format must be used for vRR.

For Junos Release 16.1 and beyond, vRR KVM bundle can be used, which already includes platform specific vRR qcow2 image: junos-x86-64-*.img. There is no need to convert the *.vmdk to *.img - as explained in this section).

The libvirt driver in the KVM/QEMU hypervisor supports qcow2-formatted images. You must convert the **vmdk** image downloaded from Juniper Networks to **qcow2** format before you can launch the vRR VM.

To launch the vRR VM:

1. Convert the **vmdk** image to **qcow2** format.

```
qemu-img convert -f vmdk -O qcow2 vmdk-filename qcow2-filename
```

For example:

```
qemu-img convert -f vmdk -O qcow2 junos-x86-64-15.1R1.9.vmdk junos-x86-64-15.1R1.9.qcow2
```

2. (For Junos OS Release 14.2 or earlier) The Routing Engine type is passed to the guest VM as an SMBIOS Type 1 argument (`product`). The libvirt versions that do not honor SMBIOS XML elements must pass this information as command line arguments. The XML configuration file used with legacy images (for Release 14.2 or earlier) must be updated with these XML elements based on the libvirt version installed on the host. Use the `libvirtd --version` command to determine the libvirt version.

   For libvirt 0.10.2 and libvirt 1.2.8, you pass the `product` argument as these command line arguments:

```
<qemu:commandline>
    <qemu:arg value='-smbios'/>
    <qemu:arg value='type=1,product=VRR'/>
</qemu:commandline>
```

   For libvirt 1.2.6, you pass the `product` argument as these XML elements:

```
<sysinfo type='smbios'>
    <system>
        <entry name='product'>VRR</entry>
    </system>
</sysinfo>
```

3. Launch the VM using the `virsh create` *`vrr-configuration-file`* command. For example:

```
virsh create vrr.xml
```

RELATED DOCUMENTATION

Configuring Interfaces, Protocols, and Routes of the Virtual Route Reflector Using Junos CLI | 17

Configuring Interfaces, Protocols, and Routes of the Virtual Route Reflector Using Junos CLI

1. Using the Junos CLI in the virtual machine console, configure the interfaces that were connected to the OVS virtual switch-br (for KVM or OpenStack) or the vSwitch (for VMware). Specify the IP addresses that were assigned to the VNICs while configuring OVS (KVM) or vSwitch (VMware).

```
[edit]
user@host# set interfaces interface-name family inet address address
user@host# set interfaces interface-name family inet6 address address
```

2. Configure the loopback interface with the IP address for the vRR.

```
[edit]
user@host# set interfaces lo0 unit 0 family inet address address
user@host# set interfaces lo0 unit 0 family inet6 address address
```

3. Add a static default route to the gateway address of the management IP address:

```
[edit]
user@host# set routing-options static route 0.0.0.0/0 next-hop address
```

4. Configure the hostname for the vRR.

```
[edit system]
user@host# set host-name hostname
```

5. Configure the root password.

```
[edit system]
user@host# set root-authentication plain-text-password
```

6. Add a user.

```
[edit system login]
user@host# set user user-name
```

7. Set the user identification (UID).

```
[edit system login user user-name]
user@host# set uid uid-value
```

8. Assign the user to a login class.

```
[edit system login user user-name]
user@host# set class class-name
```

9. Set the user password.

```
[edit system login user user-name]
user@host# set authentication plain-text-password
```

10. Enable Telnet and FTP access.

```
[edit system services]
user@host# set ftp
user@host# set telnet
```

11. Configure the types of system log messages to send to files and to user terminals.

```
[edit system syslog]
user@host# set user * any emergency
user@host# set file messages any notice
user@host# set file messages authorization info
user@host# set file interactive-commands interactive-commands any
```

12. Configure the vRR to always use 64-bit processing.

```
[edit system processes]
user@host# set routing force-64-bit
```

13. Configure the router ID and the autonomous system (AS) number.

```
[edit routing-options]
user@host# set router-id address
user@host# set autonomous-system autonomous-system
```

14. Configure BGP, including the cluster identifier and the neighbor relationships with all IBGP-enabled devices in the autonomous system (AS).

```
[edit protocols bgp group group-name]
user@host# set type internal
user@host# set local-address address
user@host# set cluster cluster-name
user@host# set neighbor address
```

15. (External peers only) Specify that the BGP next-hop value not be changed.

```
[edit protocols bgp group group-name multihop]
user@host# set no-nexthop-change
```

16. Configure a forwarding-table export policy to prevent the installation of BGP routes in the forwarding table. A vRR is not expected to be in the forwarding path for BGP service prefixes.

```
[edit policy-options]
user@host# set policy-statement policy-name term term-name from protocol bgp
user@host# set policy-statement policy-name term term-name then reject
```

17. Apply the BGP policy to the forwarding table.

```
[edit routing-options]
user@host# set forwarding-table export policy-name
```

18. Configure other desired protocols for the interfaces.

Enabling SR-IOV on vRR Instances on KVM

vRR on KVM supports single-root I/O Virtualization (SR-IOV) interface types. 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 vRR instances to improve performance.

- BIOS requirement to enable SR-IOV; Ensure that Intel VT-d or AMD IOMMU are enabled in the systems BIOS settings
- SR-IOV on VRR for KVM requires one of the following Intel NIC drivers:
  - Intel X710 or XL710 using 40G ports and i40e driver
  - Intel X520 or X540 using 10G ports and ixgbe driver
- Junos OS Release:
  - Starting with Junos OS Release 17.4 onwards, support for 40G ports with Intel X710/XL710 NICs is available on vRR.
  - Starting with Junos OS Release 20.4R3, support for 10G ports with Intel X520/X540 NICs is available on vRR.

1. Look for NICs (devices) available on your host using the **lshw -businfo -c network** command.
   - In the output below: 10G NICs are enp2s0f0 and enp2s0f1
   - 40G NICs are enp4s0f0 and enp4s0f1

   ```
   root@vrr-host: lshw -businfo -c network
   Bus info          Device      Class          Description
   ========================================================
   pci@0000:02:00.0  enp2s0f0    network        82599ES 10-Gigabit SFI/SFP+ Network Connection
   pci@0000:02:00.1  enp2s0f1    network        82599ES 10-Gigabit SFI/SFP+ Network Connection
   pci@0000:04:00.0  enp4s0f0    network        Ethernet Controller XL710 for 40GbE QSFP+
   pci@0000:04:00.1  enp4s0f1    network        Ethernet Controller XL710 for 40GbE QSFP+
   pci@0000:04:02.0  enp4s2      network        Illegal Vendor ID
   pci@0000:07:00.0  enp7s0f0    network        I350 Gigabit Network Connection
   pci@0000:07:00.1  enp7s0f1    network        I350 Gigabit Network Connection
   ```

2. Remove any VMs on the host using the device that you plan to configure SR-IOV on.

3. Check the maximum number of VFs supported on the device (for example enp2s0f0 and enp4s0f0)

```
cat /sys/class/net/ enp4s0f0/device/sriov_totalvfs 63
```

4. Configure the number of VFs you want on the device

```
echo 2 > /sys/class/net/enp4s0f0/device/sriov_numvfs (as a root user)
```

5. Verify if the VFs are created using the **lspci** command. You should see Virtual Functions corresponding to the Ethernet Controller of the device.

```
lspci | grep Ethernet
04:00.0 Ethernet controller: Intel Corporation Ethernet Controller XL710 for 40GbE QSFP+ (rev
02) - PF1
04:00.1 Ethernet controller: Intel Corporation Ethernet Controller XL710 for 40GbE QSFP+ (rev
02) - PF2
04:02.0 Ethernet controller: Intel Corporation XL710/X710 Virtual Function (rev 02) - VF1
04:0a.0 Ethernet controller: Intel Corporation XL710/X710 Virtual Function (rev 02) - VF2
```

6. Replace the existing interface stanza for the network device to obtain a new XML. Ensure to keep the management device entry as it is.

```
<interface type='hostdev' managed='yes'>
      <mac address='30:b6:4f:60:f2:82'/>  - the same as in a non-SR-IOV XML or pick a valid
mac
      <driver name='kvm'/>
      <source>
        <address type='pci' domain='0x0000' bus='0x04’ slot='0x00’ function='0x0'/> - get
this value from lspci output for the VF - column 1 domain:bus:slot:function (domain is 0x0000
by default).
      </source>
      <alias name='hostdev0'/>
      <address type='pci' domain='0x0000' bus='0x01' slot='0x01’ function='0x0'/> - keep the
same as in the existing XML
    </interface>
```

7. Bring up the VRR instance with new SRIOV based interfaces

8. Finally, to check if the VF devices that are created in Junos vRR, perform grep for “renaming” in the boot messages

```
root@device> show system boot-messages | grep renaming
re_vrr_ifd_rename: renaming ixv* -> em*  - 10G (ixv*)
re_vrr_ifd_rename: renaming ixlv* -> em* - 40G (ixlv*)
re_vrr_ifd_rename: renaming vtnet* -> em* - Management
```

RELATED DOCUMENTATION

Example: Enabling SR-IOV on vMX Instances on KVM