JUNIPER NETWORKS | Engineering Simplicity

Virtual Route Reflector

vRR Getting Started Guide for OpenStack

Published
2021-10-03

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

Virtual Route Reflector vRR Getting Started Guide for OpenStack

The information in this document is current as of the date on the title page.

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 OpenStack 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

CHAPTER 1

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**

For example, you can to upgrade vRR by using the use 64 Bit-MX High-End Series for MX240 from Downloads for MX240.

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

The following software requirements are for OpenStack.

Host OS: CentOS 7.1

- QEMU-KVM 1.5.3
- libvirt 1.2.8

Host OS: CentOS 7.2

- QEMU-KVM 1.5.3
- libvirt 1.2.17

CHAPTER 2

Installing and Configuring Virtual Route Reflector on OpenStack

Installing the Virtual Route Reflector Image Using OpenStack

IN THIS SECTION

OpenStack is a free, open-source cloud computing platform that supports creation and management of virtual Route Reflectors (vRR). OpenStack allows you to:

- Attach a VNIC to a physical NIC
- Display a graphical representation of the virtual machine
- Allocate a specific amount of disk space for the virtual machine
- Take a snapshot of a running virtual machine
- Create a new virtual machine from a snapshot

The physical and virtual machines are connected using OpenvSwitch, which eliminates the need to configure tunnels and overlays, such as MPLS over GRE and MPLS over UDP. OpenvSwitch also provides better performance than other methods, because the physical NICs are dedicated to specific VNICs.

This topic includes the following tasks:

Composing User Authentication Credentials

IN THIS SECTION

Appropriate user credentials are required for authentication to succeed for every OpenStack command. You can compose credentials with either of the following two methods:

Exporting User Credentials Once

To avoid passing user credentials every time you use the OpenStack command line, export the credentials:

1. Create a **keystonerc** file with the following contents:

```
export ADMIN_TOKEN=token
export OS_USERNAME=username
export OS_TENANT_NAME=tenant-name
export OS_PASSWORD=password
export OS_AUTH_URL=http://ip_address_of_keystone:portNumber/v2.0
```

   where

   - *token*—Authorization token
   - *username*—OpenStack user name
   - *tenant-name*—Tenant name
   - *password*—Password for OpenStack user
   - *ip_address_of_keystone*:*portNumber*—IP address of the keystone authentication server and its port number

2. Perform a sourcing of the **keystonerc** file:

```
source keystonerc
```

Passing User Credentials Each Time You Use a Command

To pass credentials on the command line every time you execute an OpenStack command:

- Enter the following in the OpenStack command line:

```
--os-username username --os-password password --os-tenant-name tenant-name --os-auth-url
http://ip_address_of_keystone:portNumber /v2.0
```

where

- *username*—OpenStack user name
- *tenant-name*—Tenant name
- *password*—Password for OpenStack user
- *ip_address_of_keystone*:*portNumber*—IP address of the keystone authentication server and its port number

Registering an Image

Before the vRR image can be used to create a vRR instance, the image must be brought into the glance directory.

To bring the vRR image into the glance directory:

- Enter the following in the OpenStack command line:

```
stack@host$ glance image-create  --name image-name --disk-format=qcow2 --container-format=
bare --file=image-location
```

where

- *image-name*—A name for the image. This name is used later when creating the vRR instance.
- *image-location*—The location of the vRR image.

Updating the Disk, CD-ROM, and VIF Settings for the Image

By default, OpenStack uses virtio for disk, CDROM, and VIF (NIC) models, but Junos-based images do not support virtio drivers. You must update the image to change these settings.

1. Enter the following in the OpenStack command line:

```
stack@host$ glance image-update --property hw_disk_bus=ide --property hw_cdrom_bus=ide --
property hw_vif_model=e1000 image-name
```

   The *image-name* is the name of the image you used in "Registering an Image" on page 8.

2. Verify that the image was brought into the glance directory by entering the following in the OpenStack command line:

```
stack@host$ glance image-list
```

   The image that you used in "Registering an Image" on page 8 should be in the list of images that is displayed.

Creating a Virtual Hardware Template

A virtual hardware template in OpenStack is called a flavor. A flavor defines a set of hardware parameters, and is later applied to the vRR instance.

To create a flavor for a vRR instance:

1. Enter the following in the OpenStack command line:

```
stack@host$ nova flavor-create --is-public true flavor-name 6 16384 10 1
```

   This makes the flavor available to the public, sets the flavor ID to 6, sets the memory size to 16384 GB, sets the disk size to 10 GB, and sets the number of virtual CPUs to one.

   The *flavor-name* is the name of the flavor, for example **VRR-flavor**.

2. Verify that the flavor was created by entering the following in the OpenStack command line:

```
stack@host$ nova flavor-list
```

The flavor that you configured in Step "1" on page 9 should be in the list of flavors that is displayed.

Creating Networks and Subnets

Create networks and subnets for internal and external communication. The virtual NICs of the vRR instances can later be attached to the subnets.

1. To create a private network, enter the following in the OpenStack command line:

```
stack@host$ neutron net-create private-network-name
```

The *private-network-name* is the name of the private network.

For example:

```
stack@host$ neutron net-create private1
```

2. To create a subnet for a private network, enter the following in the OpenStack command line:

```
stack@host$ neutron subnet-create --name private-subnet-name private-network-name subnet-cidr
```

where

- *private-subnet-name*—Name of the subnet
- *private-network-name*—Name of the private network to which the subnet belongs
- *subnet-cidr*—CIDR of the subnet

For example:

```
stack@host$ neutron subnet-create --name private1-subnet1 private1 10.0.0.0/24
```

3. To create a public network, enter the following in the OpenStack command line:

```
stack@host$ neutron net-create public-network-name --router:external=True
```

The *public-network-name* is the name of the public network.

For example:

```
stack@host$ neutron net-create public1 --router:external=True
```

4. To create a subnet for a public network, enter the following in the OpenStack command line:

```
stack@host$ neutron subnet-create public-network-name subnet-cidr --name public-subnet-name --
enable_dhcp=False --allocation-pool start=start_ip_address,end=end_ip_address  --
gateway=gateway_ip_address
```

where

- *public-network-name*—Name of the public network to which the subnet belongs
- *subnet-cidr*—CIDR of the subnet
- *public-subnet-name*—Name of the subnet
- *start_ip_address*—Lowest IP address in the allocated address range
- *end_ip_address*—Highest IP address in the allocated address range
- *gateway_ip_address*—Gateway IP address for the host machine

For example:

```
stack@host$ neutron subnet-create public1 192.168.239.90/25 --name public1-subnet1 --
enable_dhcp=False --allocation-pool start=192.168.239.64,end=192.168.239.65 --
gateway=192.168.239.126
```

5. Verify that the networks were created by entering the following in the OpenStack command line:

```
stack@host$ neutron net-list
```

The networks that you configured should be in the list that is displayed.

For example:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| id                                   | name      | subnets
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| 2d934de5-e29c-4fc0-9d00-de83dcfa2b89  | private1  | e9f89ec4-27d5-4d33-
```

```
b552-                                                    a239173bc284 10.0.0.0/24
| d1ec3880-9823-4c28-945c-2ec77b809f1a  | public1   |c65acb85-239e-4464-
add1-
a0913dab0f27                                                        192.168.239.0/25
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

6. Verify that the subnets were created by entering the following in the OpenStack command line:

```
stack@host$ neutron subnet-list
```

The subnets that you configured should be in the list that is displayed.

For example:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| id                 |name            | cidr     | allocation_pools
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| c65acb85-239e-4464- |public1-subnet1 |192.168. | {"start"; "192.168.239.64",  add1-
a0913dab0f27                     239.0/25  "end": "192.168.239.65"}
| e9f89ec4-27d5-      |private1-subnet1|10.0.0.0| {"start"; "10.0.0.2",  rd33-
b552-                              /24       "end": "192.168.239.65"}  a239173bc284
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Creating a vRR Instance

An instance is a virtual machine on which the vRR runs. To create the instance, you provide the image name, the flavor, the network ID for the virtual NIC, and a name for the instance.

To create a vRR instance:

1. Display the ID of the network that you want to associate with the vRR instance virtual NIC by entering the following in the OpenStack command line:

```
stack@host$ neutron net-list
```

For example:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| id                                   | name      | subnets
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| 2d934de5-e29c-4fc0-9d00-de83dcfa2b89  | private1  | e9f89ec4-27d5-4d33-
b552-                                                    a239173bc284 10.0.0.0/24
| d1ec3880-9823-4c28-945c-2ec77b809f1a  | public1   |c65acb85-239e-4464-
add1-
a0913dab0f27                                                          192.168.239.0/25
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

2. Record the ID of the network.
3. Enter the following in the OpenStack command line:

```
stack@host$ nova boot --image image-name flavor 6 --nic net-id=net-id "instance-name"
```

where

- *image-name*—Image name that you used in "Registering an Image" on page 8
- *net-id*—ID of the network that you want to associate with the vRR instance virtual NIC
- *instance-name*—The name for the vRR instance

For example:

```
stack@host$ nova boot --image VRR-image flavor 6 --nic net-id=2d934de5-e29c-4fc0-9d00-
de83dcfa2b89 "VRR-1"
```

4. Verify that the vRR instance has been created by entering the following in the OpenStack command line:

```
stack@host$ nova list
```

The instance that you created should be in the list that is displayed.

Creating a Router

An OpenStack router is a logical entity that routes packets among internal subnets, forwards packets from internal networks to external networks, and accesses the vRR instances from external networks. You must create a router and create an interface on the router for each subnet with which it communicates.

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

1. To create a router, enter the following in the OpenStack command line:

```
stack@host$ neutron router-create router-name
```

The *router-name* is the name for the router.

The ID of the router is displayed.

For example:

```
stack@host$ neutron router-create GWR
```

```
Created a new router:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| Field                 |Value
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| admin_state_up        |True
| external_gateway_info
| id                    | b93033e7-e825-40fa-811f-df72d3cd230d
| name                  | GWR
| status                | ACTIVE
| tenant_id             | 8d2d7bd590a14d30b4f662dbefdd8e0e
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

2. Record the ID of the router.

3. Display the ID of the subnet with which the router should communicate by entering the following in the OpenStack command line:

```
stack@host$ neutron subnet-list
```

For example:

```
stack@host$ neutron subnet-list
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| id                 |name           | cidr     | allocation_pools
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| c65acb85-239e-4464- |public1-subnet1 |192.168. | {”start”; “192.168.239.64”,  add1-
a0913dab0f27                    239.0/25  “end”: “192.168.239.65”}
| e9f89ec4-27d5-      |private1-subnet1|10.0.0.0| {”start”; “10.0.0.2”,  rd33-
b552-                             /24       “end”: “192.168.239.65”}  a239173bc284
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

4. Record the ID of the subnet.
5. Create an interface on the router for the subnet with which it communicates by entering the following in the OpenStack command line.

```
stack@host$ neutron router-interface-add router-id subnet-id
```

where

- *router-id*—ID of the router
- *subnet-id*—ID of the subnet

For example:

```
stack@host$ neutron router-interface-add b93033e7-e825-40fa-811f-df72d3cd230d e9f89ec4-27d5-
rd33-b552-a239173bc284
```

6. Display the networks.

```
stack@host$ neutron net-list
```

For example:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| id                                   | name      | subnets
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| 2d934de5-e29c-4fc0-9d00-de83dcfa2b89  | private1  | e9f89ec4-27d5-4d33-
b552-                                                    a239173bc284 10.0.0.0/24
| d1ec3880-9823-4c28-945c-2ec77b809f1a  | public1   |c65acb85-239e-4464-
add1-
a0913dab0f27                                                          192.168.239.0/25
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

7. Record the ID of the public network that should serve as the gateway for the router.
8. Configure the router as an external gateway by entering the following in the OpenStack command line.

```
stack@host$ neutron router-gateway-set router-id net-id
```

where

- *router-id*—ID of the router
- *net-id*—ID of the public network that serves as the gateway for the router

For example:

```
stack@host$ neutron router-gateway-set b93033e7-e825-40fa-811f-df72d3cd230d
d1ec3880-9823-4c28-945c-2ec77b809f1a
```

Assigning a Floating IP to a vRR Instance

A floating IP represents an external IP address, and provides access to the vRR instance from an external network. A floating IP can only be created for a network that has the `router:external` attribute.

1. Create a floating IP for the public network by entering the following in the OpenStack command line.

```
stack@host$ neutron floatingip-create public-network-name
```

The *public-network-name* is the name of the public network.

For example:

```
stack@host$ neutron floatingip-create public1
```

Information for the floating IP appears.

For example:

```
Created a new floatingip:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| Field                 |Value
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
| fixed_ip_address      |
| floating_ip_address   | 192.168.239.106
| floating_network_id   | b4934fe4-4664-4c61-b404-c6a63533e842
| id                    | aaa1fa59-7e20-4331-8d39-63c00aa29781
| port_id               |
| router_id             | 8d2d7bd590a14d30b4f662dbefdd8e0e
| status                | DOWN
| tenant_id             | e4a38502668b427c9875c591b62b76
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

2. Record the address of the floating IP.
3. Assign the floating IP to the vRR instance by entering the following in the OpenStack command line.

```
stack@host$ nova add-floating-ip instance-name floating-ip-address
```

where

- *instance-name*—Instance name that you used in "Creating a vRR Instance" on page 12
- *floating-ip-address*— IP address you recorded in Step "2" on page 17

For example:

```
stack@host$ nova add-floating-ip VRR-1 192.168.239.106
```

4. Verify that the floating IP has been created by entering the following in the OpenStack command line:

```
stack@host$ nova floating-ip-list
```

The floating IP that you created should be in the list that is displayed.

Configuring Security Group Rules

To allow access to the vRR instance via SSH and ping, you must create security rules.

1. Create a security rule for TCP traffic and assign it to the **default** security group by entering the following in the OpenStack command line.

```
stack@host$ nova secgroup-add-rule default tcp start-port-range end-port-range cidr-address-range
```

where

- *start-port-range*—Lowest port number in the allowed port range. To allow any port, use **-1 -1** for the port range.
- *end-port-range*—Highest port number in the allowed port range.
- *cidr-address-range*—CIDR of the allowed address range.

For example:

```
stack@host$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

2. Create a security rule for ICMP traffic and assign it to the **default** security group by entering the following in the OpenStack command line.

```
stack@host$ nova secgroup-add-rule default icmp start-port-range end-port-range cidr-address-range
```

For example:

```
stack@host$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

RELATED DOCUMENTATION

Configuring Interfaces, Protocols, and Routes of the Virtual Route Reflector Using Junos CLI | 19

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.