Juniper Networks | Engineering Simplicity

cRPD Deployment Guide for Linux Server

Published
2023-12-14

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

cRPD Deployment Guide for Linux Server

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 containerized routing protocol process (cRPD) in the Linux environment. This guide also includes basic cRPD container configuration and management procedures.

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

1
CHAPTER

Overview

What Is Containerized RPD?

IN THIS SECTION

The Junos® containerized routing protocol process (cRPD) is an instance of the Junos OS routing functionality instantiated inside a Linux containerized environment. cRPD provides cloud-native routing to the network. We package the cRPD software as a Docker container image. cRPD supports router functionality using IS-IS, OSPF, and BGP on the device as shown in Figure 3 on page 6.

Benefits of cRPD

- Reduced deployment time—Speed up deployment by using containers to reduce the service's boot time from several minutes to a few seconds.
- Seamless upgrade—Upgrade software with minimal service disruption.
- Flexibility—Launch multiple cRPD instances with minimum resource requirements to support the target scale.
- Stability—Provide a stable routing software on Linux.

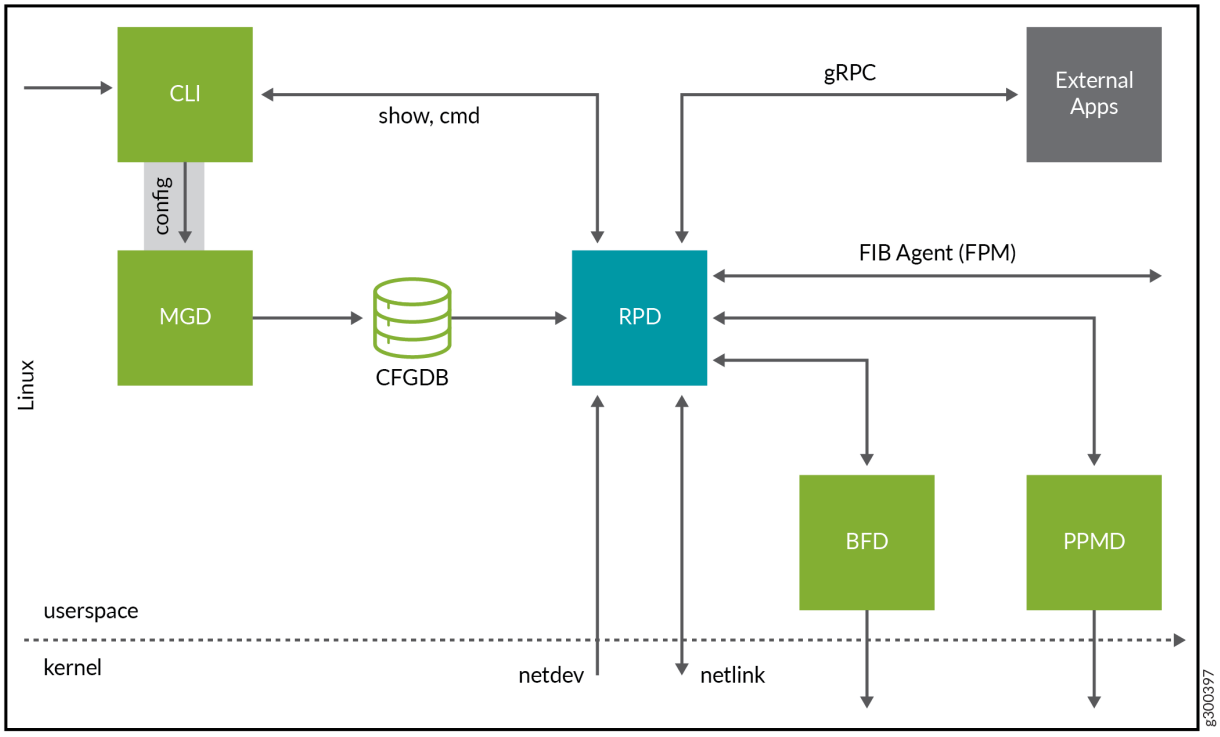
Overview of rpd on Linux

The Junos routing protocol process (rpd) is a software process within the Routing Engine software. The rpd controls the routing protocols that run on the device.

As a software process, the rpd:

- Operates from the center of a routing protocol stack based on Linux.
- Maintains one or more routing tables, which consolidate the routing information learned from all routing protocols.
- Manages all protocol messages, routing table updates, and implements routing policies.

Figure 1: RPD on Linux Architecture



You can use the rpd application to:

- Run on software containers. The cRPD application enables routing solutions such as containerized Route Reflector (cRR). The cRR service must work independently.
- Interact with mgd processes for management, CLI for configuration, BFD for detecting liveness of links, periodic packet management process (PPMD), and update protocol sessions.

- Learn the route state using various routing protocols.
- Maintain the complete set of routing information in the routing information base (RIB), also known as the routing table.
- Start all configured routing protocols and handle all routing messages. The rpd maintains one or more routing tables, which consolidate the routing information that the router learns from all routing protocols.
- Implement a routing policy with which you control the routing information that moves between the routing protocols and the routing table. Using the routing policy, you can filter and limit the transfer of information as well as set properties associated with specific routes.
- Download the routes that meet the local selection criteria into the forwarding information base (FIB), also known as the forwarding table.
- Determine the active routes for the network destinations from the routing information and program these routes into the Routing Engine's forwarding table.
- Learn the interface attributes such as names, addresses, maximum transmission unit (MTU) settings, and link status through Netlink messages.

Docker Overview

cRPD runs on any Linux distribution system that supports Docker.

Docker is an open-source software platform that you can use to create, manage, and disassemble a virtual container that can run on any Linux server. Docker packages applications in containers. You can port and use these containers on any Linux OS. A container provides OS-level virtualization for an application.

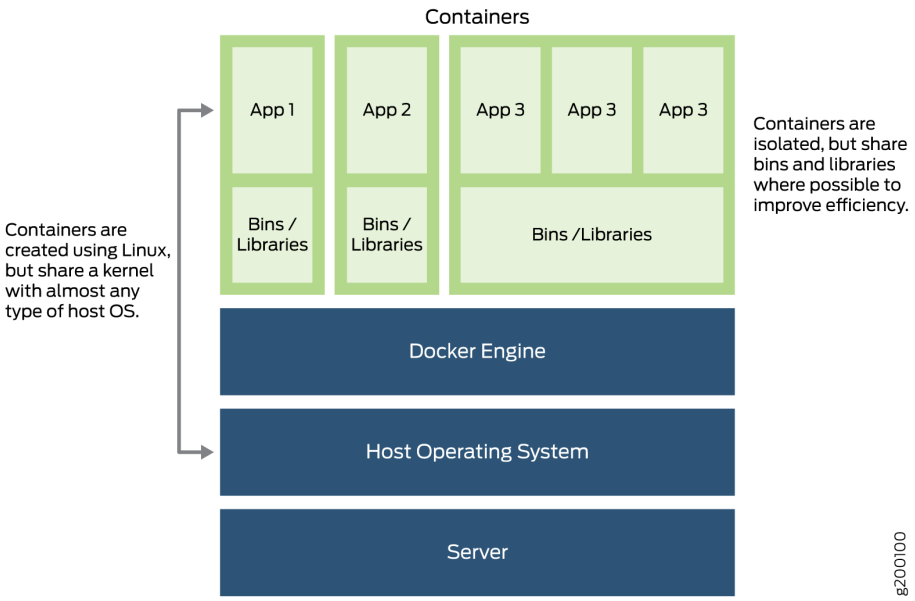
Containers don't function as virtual machines (VMs); rather they isolate virtual environments by providing them with dedicated CPU, memory, I/O, and networking capability.

Benefits of Containers

- Improved efficiency through isolation—Containers use the host OS Linux kernel features, such as the isolation of groups and namespaces, to enable multiple containers to run in isolation on the same Linux host OS. An application in a container less memory because it shares the kernel of its Linux host's OS.
- Increased spin-up (boot) speed—Containers take less time to boot as compared to VMs. Thus, you can use containers to install, run, and upgrade applications quickly and efficiently.

Figure 2 on page 5 provides an overview of a typical Docker container environment.

Figure 2: Docker Container Environment



How Does cRPD Work?

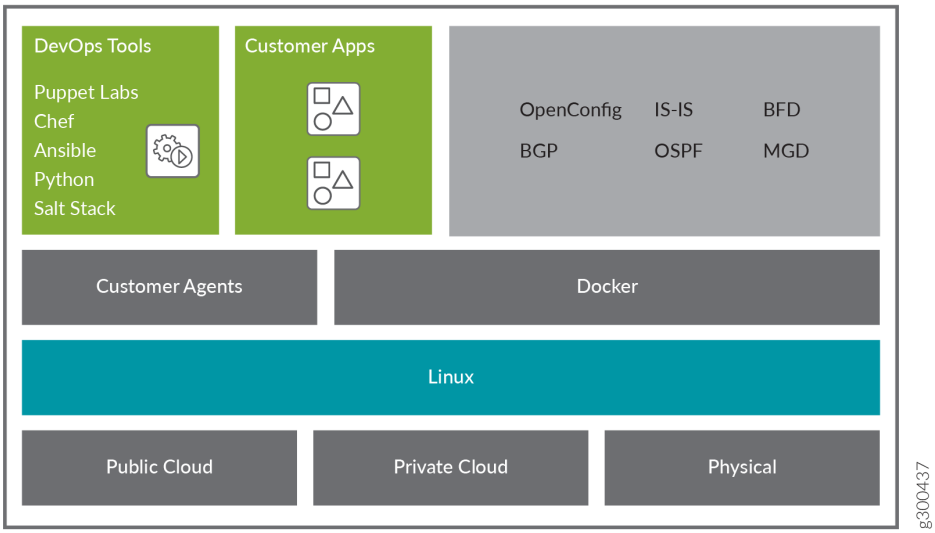
When you start Docker, a default bridge network (also called *bridge*) is created automatically, and containers connect to it unless otherwise specified. You can use this bridge network to run multiple containers on the same host without having to assign dynamic ports.

A bridge enables containers connected to the same bridge network to communicate, while providing isolation from containers that are not connected to the bridge network.

In bridge mode:

- Containers connect to the host network stack through bridges.
- Multiple containers connect to the same bridge and communicate with one another.
- The bridges enable external communication when they connect to the host OS network interfaces.

Figure 3: cRPD Overview and Functionality



When you deploy the RPD application using a container, FIB exposes the network interfaces learned by the underlying OS kernel are sent to the RPD in the Linux container. RPD learns about all the network interfaces and adds the route state for all the network interfaces. If additional Docker containers are running in the system, then all the containers and the applications running directly on the host can access the same set of network interfaces and state. cRPD forwards the routes that meet the local route selection criteria into the FIB.

Route Reflector

You can deploy cRPD to provide control plane-only services such as BGP route reflection.

cRR is hosted on a different on-network server hardware. Applications use the reachability information learned by using the route reflection service. The route reflection networking service must work independently, without depending on the same hardware or the controllers that host the applications.

Routing Engine Kernel

The Routing Engine software consists of several software processes that control router functionality and a kernel that enables communication among all the processes.

The Routing Engine kernel provides:

- Link between the routing tables and the Routing Engine's forwarding table.
- Communication with the Packet Forwarding Engine. The kernel synchronizes the Packet Forwarding Engine's copy of the forwarding table with the primary copy in the Routing Engine.

The host Linux kernel stores the FIB, where all the routes and the next-hop information are stored for packet forwarding.

The rpd runs natively on Linux and uses Netlink to share program route information with the Linux kernel. Netlink is a Linux kernel interface used for communication between the kernel and user-space processes and between different user-space processes. cRPD is an example of a user-space process.

You can use Netlink messages to:

- Program or install the FIB state generated by the rpd in the Linux kernel.
- Interact with mgd and CLI for configuration and management.
- Maintain protocol sessions using ppmd.
- Detect liveness using BFD.

Supported Features on cRPD

cRPD supports the following features:

- BGP route reflector in the Linux Containers (LXC)
- BGP add-path, multipath, graceful restart helper mode
- BGP, OSPF, OSPFv3, IS-IS, and static protocols
- BMP, BFD, and Linux FIB
- Equal-cost multipath (ECMP)
- Juniper extension toolkit (JET) for programmable RPD (PRPD)
- Junos OS CLI
- Management using open-interfaces NETCONF and SSH
- IPv4 and IPv6 routing
- MPLS routing

Licensing

You need a license to activate cRPD software features. To understand more about cRPD licenses, see Supported Features on cRPD, Flex Licenses for cRPD, and Managing cRPD Licenses.

RELATED DOCUMENTATION

- Docker Overview
- What is Docker?
- What is a Container?
- Get Started With Docker

cRPD Resource Requirements

IN THIS SECTION

Table 1 on page 8 lists the minimum resource requirements for cRPD.

Table 1: cRPD Minimum Resource Requirements

Description	Minimum Value
CPU	1 core
Memory	256 MB
Disk space	256 MB

cRPD Scaling

You can scale the performance and capacity of a cRPD by increasing the allocated amount of memory and the CPU available on the host hardware or VM resources.

Table 2 on page 9 lists the cRPD scaling information,

Table 2: cRPD Scaling

Instance	RIB/FIB Route Scale	Minimum Memory
cRPD	32,000	256 MB
	64,000	512 MB
	128,000	1024 MB
	1,000,000	2048 MB

Junos OS Features Supported on cRPD

IN THIS SECTION

Features Supported on cRPD

cRPD inherits most of the routing features with the following considerations shown in Table 3 on page 10.

Table 3: Supported Features on cRPD

Feature	Description
BGP FlowSpec	Starting in Junos OS Release 20.3R1, BGP flow specification method is supported to prevent denial -of-service attacks on the cRPD environment. [See Understanding BGP Flow Routes for Traffic Filtering.]
EVPN-VPWS	Starting in Junos OS Release 20.3R1, EVPN-VPWS is supported to provide VPWS with EVPN signaling mechanisms on cRPD. [See Overview of VPWS with EVPN Signaling Mechanisms.]
EVPN TYPE 5 with MPLS	Starting in Junos OS Release 20.3R1, EVPN Type 5 is supported for EVPN/MPLS. [See EVPN Type-5 Route with MPLS encapsulation for EVPN-MPLS.]
Segment routing	Starting in Junos OS Release 20.3R1, Segment routing support for OSPF and IS-IS protocols to provide basic functionality with Source Packet Routing in Networking (SPRING). [See Understanding Source Packet Routing in Networking (SPRING).]
Layer 2 VPN	Starting in Junos OS Release 20.3R1, support for Layer 2 circuit to provide Layer 2 VPN and VPWS with LDP signaling. [See Configuring Ethernet over MPLS (Layer 2 Circuit).]
MPLS	Starting in Junos OS Release 20.3R1, support for MPLS to provide LDP signaling protocol configuration with the control plane functionality. [See Understanding the LDP Signaling Protocol.]

Table 3: Supported Features on cRPD *(Continued)*

Feature	Description
Eventd	Starting in Junos OS Release 20.4R1, we support only external event policies. You can enable these policies in cRPD. In cRPD, eventd and rsyslogd run as independent processes. The eventd process provides eventinterface to processes such as rpd, auditd, and mgd and supports automated event policy execution. Use the `set event-options policy` *`policy name`* `events [`*`events`*`] then` command to enable an event policy and `restart event-processing` to restart event processing. By default, Python 3.x support is enabled with existing on-box Python or SLAX functions in the cRPD environment. Use the `[edit system scripts language python3]` hierarchy level to enable and to support Python event automation. [See event-options and event-policy.]
Authentication, authorization, and accounting	Starting in cRPD Release 21.1R1, you can configure local authentication, local authorization, Tacplus authentication, Tacplus authorization and Tacplus accounting at the **[edit system]** hierarchy level. We support the following features: • Local authentication and local authorization • TACACS+ authentication, authorization and accounting • User template support • Support for operational commands and regular expressions • Local authentication and remote authorization on Tacplus server. [See password-options and tacplus.]

Table 3: Supported Features on cRPD *(Continued)*

Feature	Description

SRv6 network programming in IS-IS	Starting in cRPD Release 21.1R1, you can configure to enable basic segment routing functionalities in a core IPv6 network for both route reflector role and host routing roles.
	
	You can enable SRv6 network programming in an IPv6 network at the `[edit source-packet-routing]` hierarchy level.
	
	A Segment Identifier consists of the following parts:
	
	• **Locator**— Locator is the first part of a SID that consists of the most significant bits representing the address of a particular SRv6 node. The locator is very similar to a network address that provides a route to its parent node. The IS-IS protocol installs the locator route in the `inet6.0` routing table. IS-IS routes the segment to its parent node, which subsequently performs a function defined in the other part of the SRv6 SID. You can also specify the algorithm associated with this locator.
	
	• **Function**—The other part of the SID defines a function that is performed locally on the node that is specified by the locator. There are several functions that have already been defined in the Internet draft draft-ietf-spring-srv6-network-programming-07draft, *SRv6 Network Programming*. However, we have implemented the following functions that are signalled in IS-IS. IS-IS installs these function SIDs in the `inet6.0` routing table.
	
	  • **End**— An endpoint function for SRv6 instantiation of a Prefix SID. It does not allow for decapsulation of an outer header for the removal of an SRH. Therefore, an End SID cannot be the last SID of a SID list and cannot be the Destination Address (DA) of a packet without an SRH.
	
	  • **End.X**— An endpoint X function is an SRv6 instantiation of an adjacent SID. It is a variant of the endpoint function with Layer 3 cross-connect to an array of Layer 3 adjacencies.
	
	  NOTE: The support for flavor (specifies end sid behavior) and flexible algorithm options is not available for configuring end sids.
	
	[See source-packet-routing].

Table 3: Supported Features on cRPD *(Continued)*

Feature	Description
Increase ECMP next-hop limit	Starting in cRPD Release 21.1R1, you can specify the multipath next-hop limit at the `[edit routing-options maximum-ecmp]` hierarchy level. This helps to load-balance the traffic over multiple paths. The default ECMP next-hop limit is 16. [See routing options max ecmp and "Hash Field Selection for ECMP Load Balancing on Linux " on page 90].
EVPN Type 5 with VXLAN	Starting in cRPD Release 21.1R1, we support EVPN Type 5 Route over VXLAN for both IPv4 and IPv6 prefix advertisements. [See EVPN Type-5 Route with VXLAN encapsulation for EVPN-VXLAN].
EVPN Over VXLAN Encapsulation	Starting in cRPD Release 21.2R1, we support Layer 2 EVPN Over VXLAN functionality. [See EVPN with VXLAN Data Plane Encapsulation and MAC-VRF L2 services].
Support for next-hop based dynamic tunnels	Starting in cRPD Release 21.2R1, cRPD supports to configure next-hop based dynamic IP tunnels in the Linux kernel to provide private and secure path on a public network. Whenever a tunnel needs to be installed in the kernel, a tunnel interface is created. Tunnel interfaces are created in Linux using netlink messages. The ifindex of the tunnel interface is used to listen and program the routes going over the tunnel composite next-hop. By default, MPLS-over-UDP tunnel is preferred over GRE tunnels. The following dynamic tunnels are supported: • MPLS-over-GRE (Generic Routing Encapsulation) • MPLS-over-UDP [For more information on dynamic tunnels overview, see Next-Hop-Based Dynamic Tunnels, Next-Hop Based Tunnels for Layer 3 VPNs, Configuring Next-Hop-Based MPLS-Over-UDP Dynamic Tunnels, dynamic-tunnels and Dynamic Tunnels Overview].

Table 3: Supported Features on cRPD *(Continued)*

Feature	Description
Support for SRv6 and Layer 3 services over SRv6 in BGP	Starting in cRPD Release 21.3R1, you can configure BGP based Layer 3 service over SRv6 core on cRPD. You can enable Layer 3 overlay services with BGP as control plane and SRv6 as dataplane. SRv6 network programming provides flexibility to leverage segment routing without deploying MPLS. Such networks depend only on the IPv6 headers and header extensions for transmitting data. **Limitations** • When cRPD as the PE is acting as RR, forwarding will not work using SRv6 tunnel for local PE-CE routes • Global IPv4 over SRv6 core END.DT4 is not supported with Linux kernel • Duplicate configured SRv6 SID check within a router is not supported. • SRv6 overlay service requires Service SID for forwarding. When at least one malformed SRV6 Service TLV is present in the BGP Prefix-SID attribute, instead of `treat-as-withdraw` action, the BGP update packet is ignored. On deleting `accept-srv6-service` there will not be any impact on already received routes with SRV6 SID. [For more information, see advertise-srv6-service, srv6 (BGP), Understanding SRv6 Network Programming and Layer 3 Services over SRv6 in BGP].
Support for Advanced RISC Machines (ARM64) (cRPD)	Starting in cRPD Release 21.4R1, cRPD is packaged as a docker container to run on 64-bit ARM platform. cRPD on ARM doesnot support the following features: • Sharding and updateIO. The **set system processes routing bgp rib-sharding number-of-shard** and **set system processes routing bgp update-threading number-of-threads**commands are not supported. • SRv6 [For more information, see Server Requirements].

Table 3: Supported Features on cRPD *(Continued)*

Feature	Description
Support for export of BGP Local RIB through BGP Monitoring Protocol (BMP)	Starting in cRPD Release 23.2R1, BMP is enhanced to support monitoring of local routing information base (RIB) `loc-rib` policy on cRPD. The `loc-rib` policy is added to RIB types under the `bmp route-monitoring` statement. [For more information, see Understanding the BGP Monitoring Protocol, bmp, and route-monitoring].
Interoperability of segment routing with LDP	Starting in cRPD Release 23.2R1, you can use OSPF or ISIS to enable segment routing devices to operate with the LDP devices that are not segment routing capable. [For more information, see LDP Mapping Server for Interoperability of Segment Routing and source packet routing].
Support for logging using eventd and time-zone (cRPD)	We support eventd process on cRPD to configure logging and forwarding the syslog to remote host and time zone on the system. The following support is not available on cRPD: • `help` command to view syslog information. • rsyslogd for logging. Limitations • Configuring the `management-instance` and the routing instance for the Syslog client is not supported. • TLS authentication is not supported for syslog transfer on cRPD. [For more information, see Configure Time Zones, time-zone, and "Syslog Support on cRPD" on page 62].
Support for RADIUS server (cRPD)	We provide RADIUS server support to use authentication, authorization and accounting features on cRPD. [For more information, see RADIUS Authentication, radius (System), and radius-server (System)].

Use case: Egress Peer Traffic Engineering using BGP Add-Path

Service providers, cloud operators, and enterprises can deploy Junos cRPD in their existing server-based environments to address their unique requirements.

Egress peer traffic engineering (TE) allows a central controller to instruct an ingress router in a domain.

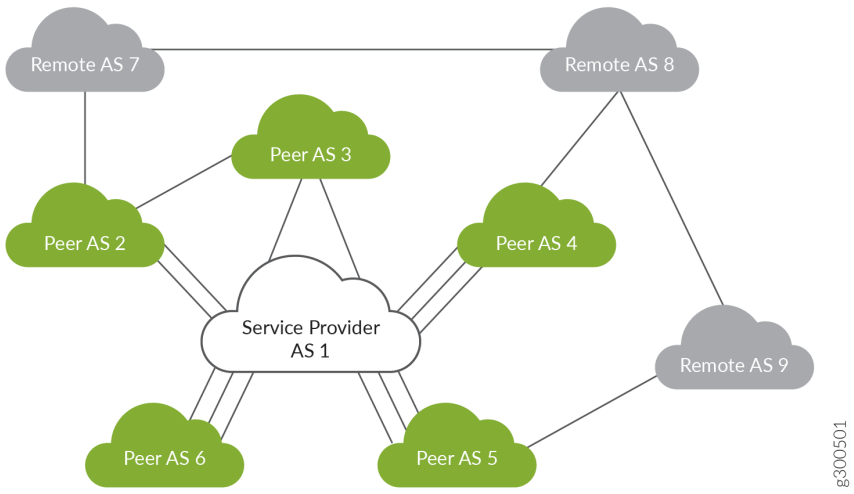
The Internet – a public global network of networks – is built as system of interconnected networks of Service Provider (SP) infrastructures. These networks are often represented as Autonomous Systems (ASs) as shown in the figure Figure 4 on page 16 each has globally unique Autonomous System Number (ASN).

The central controller directs traffic towards a specific egress router and an external interface. Thus, the traffic reaches a particular destination outside the network and optimizes utilization of the advertised egress routes.

The data-plane interconnection link (NNI) and control-plane (eBGP) direct connection between two ASs allows Internet traffic to travel between the two, usually as part of a formal agreement called peering.

A SP has multiple peering relationship with multiple other SPs. They are usually geographically distributed, differ in number and bandwidth of the NNI link, and use various business or cost models.

Figure 4: Peering Among Service Providers



In the context of AS peering, traffic egress assumes that the destination network address is reachable through a certain peer AS. So, for example, a device in Peer AS#2 can reach a destination IP address in Peer AS#4 through Service Provider AS#1.

The peer AS using an eBGP Network Layer Reachability Information (NLRI) advertisement provides the reachability information. An AS typically advertises IP addresses that belong to it, but an AS may also advertise addresses learned from another AS.

For example, Peer AS#2 can advertise addresses to SP (AS#1) that AS#2 receives from Peer AS#3, Peer AS#7, Peer AS#8, Peer AS#9, Peer AS#4 and Peer AS#5.

The reachability information advertisement depends on the BGP routing policies between the individuals ASs. Therefore, a given destination IP prefix reaches multiple peering ASs and multiple NNIs. Network administrators in the SP network select “best” exit interface for each destination prefix.

The traffic that exits the service provider AS is critical for ensuring cost efficiency while providing seamless end user experience at the same time. The definition of “best” exit interface is a combination of cost as well as latency and traffic loss.

RELATED DOCUMENTATION

- Fundamentals of Egress Peering Engineering
- BGP Labeled Unicast Egress Peer Engineering Using cRPD as Ingress

CHAPTER 2

Installing and Upgrading cRPD

Requirements for Deploying cRPD on a Linux Server

IN THIS SECTION

This section presents an overview of requirements for deploying a cRPD container on a Linux server:

Host Requirements

Table 4 on page 19 lists the Linux host requirement specifications for deploying a cRPD container on a Linux server.

Table 4: Host Requirements

Component	Specification
Linux OS support	Ubuntu 18.04 or later, RHEL 8 or later
Linux Kernel	4.8 or later
Docker Engine	18.09.1
CPUs	2 CPU core
Memory	4 GB
Disk space	10 GB
Host processor type	x86_64/ARM64 multicore CPU

Table 4: Host Requirements *(Continued)*

Component	Specification
Network Interface	Ethernet

Interface Naming and Mapping

Table 5 on page 20 lists the supported interfaces on cRPD.

Table 5: Interface Naming and Mapping

Interface Number	cRPD Interfaces
eth0	eth0-mgmt-interface
eth1	eth1-data-interface

Installing cRPD on Docker

IN THIS SECTION

This section outlines the steps to install the cRPD container in a Linux server environment that is running Ubuntu or Red Hat Enterprise Linux (RHEL). The cRPD container is packaged in a Docker image and runs in the Docker Engine on the Linux host.

This section includes the following topics:

Before You Install

Before you install cRPD as routing service to achieve routing functionality in a Linux container environment, ensure to:

- Verify the system requirement specifications for the Linux server to deploy the cRPD, see "Requirements for Deploying cRPD on a Linux Server" on page 19.

Install and Verify Docker

Install and configure Docker on Linux host platform to implement the Linux container environment, see Install Docker for installation instructions on the supported Linux host operating systems.

Verify the Docker installation. See "Debugging cRPD Application" on page 141.

To install the latest Docker:

Log in and download the software.

```
root@ubuntu-vm18:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

root@ubuntu-vm18:~# add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $
(lsb_release -cs) stable"

root@ubuntu-vm18:~# apt update

root@ubuntu-vm18:~# apt install docker-ce
```

Download the cRPD Software

The cRPD software is available as a cRPD Docker file from the Juniper Internal Docker registry.

There are two ways to download the software:

- Juniper Docker Registry

- Juniper software download page

Prerequisites

- Before you import the cRPD software, ensure that Docker is installed on the Linux host and that the Docker Engine is running.
- Ensure to register with Juniper Support before you download the cRPD software.

Once the Docker Engine has been installed on the host, perform the following to download and start using the cRPD image:

To download the cRPD software using the Juniper Docker Registry:

1. Log in to the Juniper Internal Docker registry using the following steps:
   a. Create the file **-passwd.txt** and copy the access token excluding the quotes provided by Juniper customer care team.
   b. Login to enterprise hub using the following command:

      `root@ubuntu-vm18$` **cat passwd.txt | docker login -u"<registered-email-id>" --password-stdin enterprise-hub.juniper.net:443**

      For example, `root@ubuntu-vm18$` **cat passwd.txt | docker login -u user@domain.com --password-stdin enterprise-hub.juniper.net:443**
2. Pull the docker image from the download site using the following command:

   `root@ubuntu-vm18:~#` **docker pull enterprise-hub.juniper.net:443/crpd-docker-prod/crpd:<release tag>**

   `root@ubuntu-vm18:~#` **docker pull enterprise-hub.juniper.net:443/crpd-docker-prod/crpd:22.3R1**
3. Verify images in docker image repository.

   `root@ubuntu-vm18:~#` **docker images**

```
REPOSITORY
TAG                             IMAGE ID       CREATED         SIZE
enterprise-hub.juniper.net:443/crpd-docker-prod/crpd
latest                          5d3c29ee4521   3 months ago    550MB
enterprise-hub.juniper.net:443/crpd-docker-prod/crpd
22.3R1                          5dfdda6ea2de   5 months ago    461MB
```

Download cRPD Software from Juniper URL

To download the cRPD software from the Juniper download URL:

1. Download the cRPD software image from the Juniper Networks website.`root@ubuntu-vm18:~#` **docker load -i junos-routing-crpd-docker-19.2R1.8.tgz**
2. Verify the downloaded images in docker image repository.`root@ubuntu-vm18:~#` **docker images**

```
REPOSITORY              TAG                             IMAGE ID
CREATED
SIZE

crpd                    19.2R1.8                        4156a807054a        6 days
ago          278MB
```

Creating Data Volumes and Running cRPD using Docker

To create data volumes:

1. Create data volume for configuration and var logs.

   `root@ubuntu-vm18:~#` **docker volume create crpd01-config**

   ```
   crpd01-config
   ```

   `root@ubuntu-vm18:~#` **docker volume create crpd01-varlog**

   ```
   crpd01-varlog
   ```

   Data volumes remain even after containers are destroyed and can be attached to newer containers. Data volumes are not shared between multiple containers at the same time unless they are ready-only volumes.
2. Download and load the cRPD software.
3. Attach the data volumes to create and launch the container to the cRPD instance.

   In the bridge mode, containers are connected to host network stack through bridge(s). Multiple containers can connect to the same bridge and communicate with each other. External devices communication is possible, if the bridge is connected to the host OS network interfaces.

For routing purposes, it is also possible to assign exclsuively assign all or a subset of physical interfaces for exclusive use by a docker container.

> **NOTE**: You must include the `--privileged` flag in the `docker run` command to enable the cRPD container to run in privileged mode.

`root@ubuntu-vm18:~#` **docker run --rm --detach --name crpd01 -h crpd01 --net=bridge --privileged -v crpd01-config:/config -v crpd01-varlog:/var/log -it enterprise-hub.juniper.net/crpd-docker-prod/crpd:19.2R1.8**

Bridge mode is the default working mode of docker. This allows to run multiple containers to run on same host without any assignment of dynamic port. Each container runs its own private network namespace.

To launch cRPD in host networking mode:

1. In the host mode, the network namespace is shared. For example, if an interface is defined inside a pod, the same interface is visible on the host as well. Docker containers uses the host network namespace. Run the command to launch cRPD in host networking mode:

   `root@ubuntu-vm18:~#` **docker run --rm --detach --name crpd01 -h crpd01 --privileged --net=host -v crpd01-config:/config -v crpd01-varlog:/var/log -it crpd:19.2R1.8**

Configuring Memory

To limit the amount of memory allocated to the cRPD:

You can specify the memory size using the following command:

`root@ubuntu-vm18:~#` **docker run --rm --detach --name crpd01 -h crpd01 --privileged -v crpd01-config:/config -v crpd01-varlog:/var/log -m 2048MB --memory-swap=2048MB -it crpd:19.2R1.8**

Configuring cRPD using the CLI

cRPD provides Junos command line configuration and operational commands for routing service. It provides subset of routing protocols configuration that enable node participates in topology and routing.

You can configure interfaces from Linux shell. Interface configuration is available only for the ISO addresses.

To configure the cRPD container using the CLI:

1. Log in to the cRPD container.

   root@ubuntu-vm18:~/# **docker exec -it crpd01 cli**

2. Enter configuration mode.

   root@crpd01> **configure**

```
Entering configuration mode
[edit]
```

3. Set the root authentication password by entering a cleartext password, an encrypted password, or an SSH public key string (DSA or RSA).

   root@crpd01# **set system root-authentication plain-text-password**

```
New password: password
Retype new password: password
```

4. Commit the configuration to activate it on the cRPD instance.

   root@crpd01# **commit**

```
commit complete
```

5. (Optional) Use the show command to display the configuration to verify that it is correct.

   root@crpd01# **show**

```
## Last changed: 2019-02-13 19:28:26 UTC
version "19.2I20190125_1733_rbu-builder [rbu-builder]";
system {
    root-authentication {
        encrypted-password "$6$JEc/p
$QOUpqi2ew4tVJNKXZYiCKT8CjnlP3SLu16BRIxvtz0CyBMc57WGu2oCyg/lTr0iR8oJMDumtEKi0HVo2NNFEJ."; ##
SECRET-DATA
    }
}
```

RELATED DOCUMENTATION

- Docker Engine User Guide
- Docker Commands

Installing cRPD on Kubernetes

IN THIS SECTION

Kubernetes is a open-source platform for managing containerized workloads and services. Containers are a good way to bundle and run the applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Kubernetes provides you with a framework to run distributed systems resiliently. Kubernetes provides a platform for deployment automation, scaling, and operations of application containers across clusters of host containers.

Prerequisite

Install Kubernetes on Linux system and also to deploy Kubernetes on a two-node Linux cluster, see Kubernetes Installation.

When you deploy Kubernetes, you get a cluster. A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node. The worker node(s) host the pods that are the components of the application.

This section outlines the steps to create the cRPD Docker image on Kubernetes.

Installing Kubernetes

To install Kubernetes:

1. Login as root user.
2. Download and install the software.

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/
kubernetes.list
```

```
apt-get update
```

```
apt-get install -y kubectl
```

```
wget https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

```
cp minikube-linux-amd64 /usr/local/bin/minikube
```

```
chmod 755 /usr/local/bin/minikube
```

3. Start Kubernetes.

```
apt install conntrack
```

```
root@crpd-01:~# minikube start --driver=none
```

```
* minikube v1.15.1 on Ubuntu 18.04
```

```
* Automatically selected the docker driver
...
```

Kubernetes Cluster

Kubernetes coordinates a cluster of computers that are connected to work as a single unit. Kubernetes automates the deployment and scheduling of cRPD across a cluster in an efficient way.

A Kubernetes cluster consists of two types of resources:

- The Primary coordinates the cluster
- Nodes are the workers that run applications

The Primary is responsible for managing the cluster. The primary coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster. Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes master. The node should also have tools for handling container operations, such as Docker or rkt. A Kubernetes cluster that handles production traffic should have a minimum of three nodes.

When you deploy cRPD on Kubernetes, the primary starts the application containers. The primary schedules the containers to run on the cluster's nodes. The nodes communicate with the primary using the Kubernetes API, which the primary exposes. End users can also use the Kubernetes API directly to interact with the cluster.

A Pod always runs on a Node. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Primary. A Node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster.

Every Kubernetes Node runs at least:

- Kubelet, a process responsible for communication between the Kubernetes Master and the Node; it manages the Pods and the containers running on a machine.
- A container runtime (like Docker, rkt) responsible for pulling the container image from a registry, unpacking the container, and running the application.

To create minikube cluster:

1. Run the following command to verify the minikube version:

```
minikube version
```

2. Run the following command to start the cluster:

```
minikube start
```

3. Run the following command to verify if kubectl is installed:

```
kubectl version
```

4. Run the following command to view the cluster details:

```
kubectl cluster-info
```

5. Run the following command to view the nodes in the cluster:

```
kubectl get nodes
```

Download cRPD Docker Image

Prerequisites

- Before you import the cRPD software, ensure that Docker is installed on the Linux host and that the Docker Engine is running.
- Ensure to register with Juniper Support before you download the cRPD software.

To download the docker image:

1. Log in to the Juniper Internal Docker registry using the login name and password that you received as part of the sales fulfillment process when ordering cRPD.
   a. Create the file **-passwd.txt** and copy the access token excluding the quotes provided by Juniper customer care team.
   b. Login to enterprise hub using the following command:

      root@ubuntu-vm18$ **cat passwd.txt | docker login -u"<registered-email-id>" --password-stdin enterprise-hub.juniper.net:443**

For example, root@ubuntu-vm18$ **cat passwd.txt | docker login -u user@domain.com --password-stdin enterprise-hub.juniper.net:443**

2. Pull the docker image from the download site using the following command:

   root@dc-rpd-01# **docker pull enterprise-hub.juniper.net:443/crpd-docker-prod/crpd:<release tag>**

   For example,

   root@ubuntu-vm18:~# **docker pull enterprise-hub.juniper.net:443/crpd-docker-prod/crpd:22.3R1**

3. Verify images in docker image repository.

   root@dc-rpd-01# **docker images**

```
REPOSITORY
TAG                             IMAGE ID       CREATED         SIZE
enterprise-hub.juniper.net:443/crpd-docker-prod/crpd
latest                          5d3c29ee4521   3 months ago    550MB
enterprise-hub.juniper.net:443/crpd-docker-prod/crpd
22.3R1                          5dfdda6ea2de   5 months ago    461MB
```

Creating a cRPD Pod using Deployment

A Kubernetes Pod is a group of one or more Containers, tied together for the purposes of administration and networking. A Kubernetes Deployment checks on the health of your Pod and restarts the Pod's Container if it terminates. Deployments are the recommended way to manage the creation and scaling of Pods.

When you describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

1. Create the crpd.yaml file on Kubernetes-master and add the following text content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
   name: crpd
   namespace: default
   labels:
```

```
      app: crpd
spec:
   selector:
     matchLabels:
       app: crpd
   template:
     metadata:
       labels:
         app: crpd
       annotations:
         deployment.kubernetes.io/revision: "1"
       generation: 2
     spec:
       containers:
       - name: crpd
         image: "enterprise-hub.juniper.net/crpd-docker-prod/crpd:20.1R1.11"
         imagePullPolicy: ""
       imagePullSecrets:
       - name: routing-registry
```

2. Save the crpd.yaml file to create the cRPD Pod.

   root@kubernetes-master:~# **kubectl create -f crpd.yaml**

```
deployment.apps/crpd created
```

3. Run the following command to view the list of existing Pods:

   root@kubernetes-master:~# **kubectl get pods**

```
NAME                    READY   STATUS    RESTARTS   AGE
crpd-5fc4fd79df-579cn   1/1     Running   0          71s
```

4. Run the following command to view what containers are inside that Pod and what images are used to build the containers:

   root@kubernetes-master:~# **kubectl describe pod crpd**

```
Name:         crpd-5fc4fd79df-579cn
Namespace:    default
Priority:     0
Node:         ix-crpd-01/10.102.70.153
```

```
Start Time:   Mon, 28 Dec 2020 19:12:33 +0000
Labels:       app=crpd
              pod-template-hash=5fc4fd79df
Annotations:  deployment.kubernetes.io/revision: 1
Status:       Running
IP:           172.17.0.3
IPs:
  IP:           172.17.0.3
Controlled By:  ReplicaSet/crpd-5fc4fd79df
Containers:
  crpd:
    Container ID:   docker://a211f6993f9ab5793c3de5aaef2c97ba600b991bebd801f947f179e8e915a323
    Image:          enterprise-hub.juniper.net/crpd-docker-prod/crpd:20.1R1.11
    Image ID:       docker-pullable://enterprise-hub.juniper.net/crpd-docker-prod/
crpd@sha256:1e82c06654caf47aa22e4a020b8bea02562fa25ba7abe80affab998199ae69ab
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 28 Dec 2020 19:12:55 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-xcpf7 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-xcpf7:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-xcpf7
    Optional:    false
QoS Class:       BestEffort
 Node-Selectors:  <none>
 Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                  node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  97s   default-scheduler  Successfully assigned default/
```

```
crpd-5fc4fd79df-579cn to ix-crpd-01
  Normal  Pulled     79s   kubelet            Container image "enterprise-hub.juniper.net/
crpd-docker-prod/crpd:20.1R1.11" already present on machine
  Normal  Created    74s   kubelet            Created container crpd
  Normal  Started    72s   kubelet            Started container crpd
```

Creating a cRPD Pod using YAML

A Pod is the basic execution unit of a Kubernetes application–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents a unit of deployment: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources. Docker is the most common container runtime used in a Kubernetes Pod.

You can directly create a Pod or indirectly using a Controller in Kubernetes. A Controller can create and manage multiple Pods. Controllers use a Pod template that you provide to create the Pods. Pod templates are pod specifications which are included in other objects, such as Replication Controllers, Jobs, and DaemonSets.

To create the cRPD pod using the YAML file

1. Create the crpd.yaml file on Kubernetes-master add the following text content:

```
------------------
apiVersion: v1
kind: Pod
metadata:
  name: crpd
  labels:
    app: crpd
spec:
  containers:
  - name: crpd
    image: "enterprise-hub.juniper.net/crpd-docker-prod/crpd:20.1R1.11"
    imagePullPolicy: ""
    ports:
    - containerPort: 179
    securityContext:
      privileged: true
```

```
-----------------
```

2. Save the crpd.yaml file to create the crpd Pod.

   root@kubernetes-master:~# **kubectl create -f crpd.yaml**

```
pod/crpd created
```

3. Run the following command to view the list of existing Pods:

   root@kubernetes-master:~# **kubectl get pods**

```
NAME                    READY   STATUS      RESTARTS   AGE
crpd                    1/1     Running     0          18h
crpd-5fc4fd79df-xr8f5   1/1     Running     0          19h
```

4. Run the following command to view what containers are inside that Pod and what images are used to build the containers:

   root@kubernetes-master:~# **kubectl describe pod crpd**

```
Name:               crpd
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               kubernetes-slave/10.102.183.130
Start Time:         Thu, 09 May 2019 13:38:33 -0700
Labels:             app=crpd
Annotations:        <none>
Status:             Running
IP:                 10.244.1.9
Containers:
  crpd:
    Container ID:   docker://cc7e9187ad2d482420b3afc90843443a318abb3f354da6bf2da5d4c5f7b791cc
    Image:          crpd
    Image ID:       docker://
sha256:1bc4b1c99f81f7d88b73a04f9426e360ae2cc9ea0330442e633a6ebdfec00af0
    Ports:          179/TCP
    Host Ports:     0/TCP, 0/TCP
    State:          Running
```

```
      Started:      Thu, 09 May 2019 13:38:36 -0700
    Ready:          True
    Restart Count:  0
        Environment:    <none>
        Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-ncktj (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-ncktj:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-ncktj
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:   <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From                       Message
  ----    ------     ----  ----                       -------
  Normal  Pulled     60s   kubelet, kubernetes-slave  Container image "crpd-ui32" already
present on machine
  Normal  Created    60s   kubelet, kubernetes-slave  Created container crpd
  Normal  Started    59s   kubelet, kubernetes-slave  Started container crpd
  Normal  Scheduled  54s   default-scheduler          Successfully assigned default/crpd to
kubernetes-slave
```

5. Run the following command to provide an interactive CLI inside the running container:

   `root@kubernetes-master:~#` **kubectl exec -it crpd cli**Here you are running a pod with the name crpd and connect to the command line mode.

```
===>
           Containerized Routing Protocols Daemon (CRPD)
 Copyright (C) 2018-19, Juniper Networks, Inc. All rights reserved.
                                                                  <===
```

6. Run the following command to view the routes:

root@crpd:/> **show route**

```
inet.0: 2 destinations, 2 routes (2 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

172.17.0.0/16      *[Direct/0] 00:04:31
                    >  via eth0
172.17.0.3/32      *[Local/0] 00:04:31
                       Local via eth0

inet6.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

fe80::42:acff:fe11:3/128
                   *[Local/0] 00:04:31
                       Local via eth0
fe80::5873:acff:feee:fc29/128
                   *[Local/0] 00:04:31
                       Local via lsi
ff02::2/128        *[INET6/0] 00:04:31
                       MultiRecv
```

Each Pod is meant to run a single instance of a given application. If you want to scale your application horizontally (e.g., run multiple instances), you should use multiple Pods, one for each instance. In Kubernetes, this is generally referred to as replication.

SEE ALSO

Deployments

Creating a cRPD Pod using Job Resource

A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate. When a specified number of successful completions is reached, the task is complete. You can also use a Job to run multiple Pods in parallel. Deleting a Job will clean up the Pods it created. Suspending a Job will delete its active Pods until the Job is resumed again. To create the cRPD Pod using the crpd_job.yaml file:

1. Create the crpd_job.yaml file on work nodes and add the following text content:

```
------------------
apiVersion: batch/v1
kind: Job
metadata:
  name: crpdjob
spec:
  template:
    spec:
      containers:
      - name: crpdjob
        image: "enterprise-hub.juniper.net/crpd-docker-prod/crpd:20.1R1.11"
        imagePullPolicy: ""
        command: ["/bin/bash"]
      restartPolicy: Never
  backoffLimit: 4------------------
```

2. Save the crpd_job.yaml file to create the crpd Pod.

```
root@kubernetes-master:~# kubectl create -f crpd_job.yaml
```

```
job.batch/crpdjob created
```

3. Run the following command to view the list of existing Pods:

```
root@kubernetes-master:~# kubectl get pods
```

```
NAME                    READY   STATUS      RESTARTS   AGE
crpd                    1/1     Running     0          18h
crpd-5fc4fd79df-xr8f5   1/1     Running     0          19h
crpdjob-kdgg4           0/1     Completed   0          19h
```

4. Run the following command to view what containers are inside that Pod and what images are used to build the containers:

```
root@kubernetes-master:~# kubectl describe job/crpdjob
```

```
Name:           crpdjob
Namespace:      default
Selector:       controller-uid=5997b1fe-d899-4c59-bffb-4cd9a1d72d8b
```

```
Labels:         controller-uid=5997b1fe-d899-4c59-bffb-4cd9a1d72d8b
                job-name=crpdjob
Annotations:    <none>
Parallelism:    1
Completions:    1
Start Time:     Tue, 22 Oct 2019 02:34:25 -0700
Completed At:   Tue, 22 Oct 2019 02:34:27 -0700
Duration:       2s
Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=5997b1fe-d899-4c59-bffb-4cd9a1d72d8b
           job-name=crpdjob
  Containers:
   crpdjob:
    Image:      crpd:19.2R1.8
    Port:       <none>
    Host Port:  <none>
    Command:
      /bin/bash
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age    From            Message
  ----    ------            ----   ----            -------
  Normal  SuccessfulCreate  4m36s  job-controller  Created pod: crpdjob-9lspk
```

Creating a cRPD Pod using DaemonSet

DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

Creating the cRPD pod using the crpd_daemonset.yaml file

1. Create the crpd_daemonset.yaml file on work nodes and add the following text content:

```
------------------
apiVersion: apps/v1
kind: DaemonSet
```

```
metadata:
  name: crpdda1
  labels:
    app: crpdda
spec:
  selector:
    matchLabels:
      app: crpdda
  template:
    metadata:
      labels:
        app: crpdda
    spec:
      hostNetwork: true
      nodeSelector:
        disktype: ssd
      containers:
        - name: crpd
          image: crpd:19.2R1.8
          imagePullPolicy: Never
          ports:
          - containerPort: 179
          - containerPort: 40051

------------------
```

2. Save the da1.yaml file to create the crpd Pod.

root@kubernetes-master:~# **kubectl create -f crpd_daemonset.yaml**

```
daemonset.apps/crpdda1 created
```

3. Run the following command to view the list of existing Pods:

root@kubernetes-master:~# **kubectl get pods**

```
NAME                   READY   STATUS      RESTARTS   AGE
crpd                   1/1     Running     0          18h
crpd-5fc4fd79df-xr8f5  1/1     Running     0          19h
crpdjob-kdgg4          0/1     Completed   0          19h
```

4. Run the following command to view what containers are inside that Pod and what images are used to build the containers:

root@kubernetes-master:~# **kubectl describe pod crpd**

```
Name:               crpd
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               kubernetes-slave1/10.49.107.224
Start Time:         Wed, 16 Oct 2019 23:20:49 -0700
Labels:             app=crpdda
                    controller-revision-hash=5d88785f7c
                    pod-template-generation=1
Annotations:        <none>
Status:             Running
IP:                 10.244.1.19
Controlled By:      DaemonSet/crpdda1
Containers:
  crpd:
    Container ID:   docker://0e13bdaa97c4a6da46c2fe3008939652031633d44440699ce71f094763a40244
    Image:          crpd:19.2R1.8
    Image ID:       docker://
sha256:8e00d0d60309cd0d0bee63fea865b1e389f803a57b1239386e03b31a01146dbf
    Ports:          179/TCP, 40051/TCP
    Host Ports:     0/TCP, 0/TCP
    State:          Running
      Started:      Wed, 16 Oct 2019 23:20:51 -0700
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-5chxw (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-5chxw:
    Type:        Secret (a volume populated by a Secret)
```

```
    SecretName:  default-token-5chxw
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/disk-pressure:NoSchedule
                 node.kubernetes.io/memory-pressure:NoSchedule
                 node.kubernetes.io/not-ready:NoExecute
                 node.kubernetes.io/pid-pressure:NoSchedule
                 node.kubernetes.io/unreachable:NoExecute
                 node.kubernetes.io/unschedulable:NoSchedule
Events:
  Type    Reason     Age        From                         Message
  ----    ------     ----       ----                         -------
  Normal  Scheduled  <unknown>  default-scheduler            Successfully assigned default/
crpdda1-tr85h to kubernetes-slave1
  Normal  Pulled     25s        kubelet, kubernetes-slave1  Container image "crpd:19.2R1.8"
already present on machine
  Normal  Created    24s        kubelet, kubernetes-slave1  Created container crpd
  Normal  Started    24s        kubelet, kubernetes-slave1  Started container crpd
```

root@kubernetes-master:~# **kubectl describe pod crpd-5fc4fd79df-xr8f5**

```
Name:               crpd-5fc4fd79df-xr8f5
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               kubernetes-slave2/10.49.107.220
Start Time:         Wed, 16 Oct 2019 23:20:53 -0700
Labels:             app=crpdda
                    controller-revision-hash=5d88785f7c
                    pod-template-generation=1
Annotations:        <none>
Status:             Running
IP:                 10.244.2.19
Controlled By:      DaemonSet/crpdda1
Containers:
  crpd:
    Container ID:   docker://296e68358a6b85a92216954b1f703315eebd2b21f3ffacf91d0f197ab7da21ee
    Image:          crpd:19.2R1.8
    Image ID:       docker://
sha256:82d848c70c24b225fc2ebfa6c39c123153ddde7e3bd5ed40876f537c02693047
```

```
    Ports:          179/TCP, 40051/TCP
    Host Ports:     0/TCP, 0/TCP
    State:          Running
      Started:      Wed, 16 Oct 2019 23:20:56 -0700
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-5chxw (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-5chxw:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-5chxw
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/disk-pressure:NoSchedule
                 node.kubernetes.io/memory-pressure:NoSchedule
                 node.kubernetes.io/not-ready:NoExecute
                 node.kubernetes.io/pid-pressure:NoSchedule
                 node.kubernetes.io/unreachable:NoExecute
                 node.kubernetes.io/unschedulable:NoSchedule
Events:
  Type    Reason     Age        From                          Message
  ----    ------     ----       ----                          -------
  Normal  Scheduled  <unknown>  default-scheduler             Successfully assigned default/
crpdda1-v48lc to kubernetes-slave2
  Normal  Pulled     30s        kubelet, kubernetes-slave2  Container image "crpd:19.2R1.8"
already present on machine
  Normal  Created    30s        kubelet, kubernetes-slave2  Created container crpd
  Normal  Started    29s        kubelet, kubernetes-slave2  Started container crpd
```

Each Pod is meant to run a single instance of a given application. If you want to scale your application horizontally (e.g., run multiple instances), you should use multiple Pods, one for each instance. In Kubernetes, this is generally referred to as replication.

SEE ALSO

DaemonSet

Scaling of cRPD

You can create multiple instances of cRPD based on the demand using the `-replicas` parameter for the `kubectl run` command. Deployment is an object which can own and manage their ReplicaSets.

We should have one pod existing before scaling.

To scale up:

1. Create the Pod.

   `root@kubernetes-master:~#` **kubectl create -f crpd_replicatset.yaml**

```
deployment.apps/crpdref created
```

2. Create the crpd_replicaSet.yaml file on Kubernetes-master add the following text content:

```
------------------
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  generation: 2
  labels:
    run: reflector
  name: crpdref
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      run: reflector
  template:
    metadata:
      labels:
        run: reflector
    spec:
```

```
      containers:
      - name: reflector
        image: "enterprise-hub.juniper.net/crpd-docker-prod/crpd:20.1R1.11"
        imagePullPolicy: ""
-----------------
```

3. Run the following command to view the Pods:

   root@kubernetes-master:~# **kubectl get pods**

```
NAME                       READY   STATUS    RESTARTS   AGE
crpdref-76c5c7b884-hxbtr   1/1     Running   0          10s
crpdref-76c5c7b884-wp55c   1/1     Running   0          10s
```

4. Run the following command to scale the Deployment to 4 replicas:

   root@kubernetes-master:~# **kubectl scale deployments crpdref --replicas=4**

```
deployment.apps/crpdref scaled
```

5. Run the following command to list the deployments:

   root@kubernetes-master:~# **kubectl get deployments**

6. Run the following command to check the number of pods changed:

   root@kubernetes-master:~# **kubectl get pods -o wide**

```
NAME                       READY   STATUS    RESTARTS   AGE     IP            NODE
NOMINATED NODE   READINESS GATES
crpd                       1/1     Running   2          3h35m   172.17.0.2    ix-crpd-01
<none>           <none>
crpd-5fc4fd79df-8vxtp      1/1     Running   0          5m43s   172.17.0.9    ix-crpd-01
<none>           <none>
crpd-5fc4fd79df-cd5g8      1/1     Running   0          5m43s   172.17.0.8    ix-crpd-01
<none>           <none>
crpd-5fc4fd79df-hmm5q      1/1     Running   0          5m43s   172.17.0.7    ix-crpd-01
<none>           <none>
crpd-5fc4fd79df-xr8f5      1/1     Running   2          23h     172.17.0.3    ix-crpd-01
<none>           <none>
crpdref-76c5c7b884-dbg5p   1/1     Running   0          5m12s   172.17.0.11   ix-crpd-01
<none>           <none>
```

```
crpdref-76c5c7b884-h8xks   1/1     Running   0          5m12s   172.17.0.10   ix-crpd-01
<none>           <none>
```

7. Run the following command to check the details of the Pods:

   root@kubernetes-master:~# **kubectl describe pods**

To scale down:

1. Run the following command to scale down the Service to 2 replicas:

   root@kubernetes-master:~# **kubectl scale deployments crpdref --replicas=2**

   ```
   deployment.apps/crpdref scaled
   ```

2. Run the following command to list the deployments:

   root@kubernetes-master:~# **kubectl get deployments**

3. Run the following command to list the number of Pods. You can view the 2 Pods were terminated:

   root@kubernetes-master:~# **kubectl get pods -o wide**

Rolling Update of cRPD Deployment

You can update Pod instances with new versions. Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods are scheduled on Nodes with available resources. Rollback updates promotes an application from one environment to another with continuous integration and continuous delivery of applications with zero downtime. In Kubernetes, updates are versioned and any Deployment update can be reverted to previous stable version.

To update cRPD deployment with new image and preserve the configuration after update:

1. Create the cRPD Pod.

   root@crpd-01:~# **kubectl kubectl create -f crpd_deploy.yaml**

   ```
   deployment.apps/crpd-deploy created
   ```

2. Create the crpd_deploy.yaml file on Kubernetes-master add the following text content:

```
-------------------
apiVersion: apps/v1
kind: Deployment
metadata:
  name: crpd-deploy
  labels:
    app: crpd
spec:
  replicas: 2
  selector:
    matchLabels:
      app: crpd
  template:
    metadata:
      labels:
        app: crpd
    spec:
      containers:
      - name: crpd
        image: "enterprise-hub.juniper.net/crpd-docker-prod/crpd:19.4R1.10"
        imagePullPolicy: ""
        ports:
        - containerPort: 179
        - containerPort: 40051------------------
```

3. Run the following command to list the deployments:

```
root@kubernetes-master:~# kubectl get deployments
```

4. Run the following command to list the running pods:

```
root@kubernetes-master:~# kubectl get pods
```

```
NAME                           READY   STATUS    RESTARTS   AGE
...
crpd-deploy-6c489b5b8b-4tqgn   1/1     Running   0          9m20s
crpd-deploy-6c489b5b8b-vp7df   1/1     Running   0          9m21s
```

```
root@kubernetes-master:~# kubectl get pods -o wide
```

```
NAME                            READY   STATUS    RESTARTS   AGE     IP
NODE         NOMINATED NODE   READINESS GATES
...
crpd-deploy-6c489b5b8b-4tqgn   1/1     Running   0          6m57s   172.17.0.10   ix-
crpd-01   <none>           <none>
crpd-deploy-6c489b5b8b-vp7df   1/1     Running   0          6m58s   172.17.0.11   ix-
crpd-01   <none>           <none>
```

5. Run the following command to view the current image version of the cRPD:

```
root@kubernetes-master:~# kubectl exec -it crpd-deploy4-674b4fcfb5-8xc5d -- cli
```

```
===>
           Containerized Routing Protocols Daemon (CRPD)
 Copyright (C) 2018-19, Juniper Networks, Inc. All rights reserved.
                                                                  <===
```

6. Run the following command to view the current image version:

```
root@crpd-deploy4-674b4fcfb5-8xc5d> show version
```

```
Hostname: crpd-deploy4-674b4fcfb5-8xc5d
Model: cRPD
Junos: 20.4R1.12
cRPD package version : 20.4R1.12 built by builder on 2020-12-20 13:35:15 UTC
```

7. Run the following command to update the image of the application to new version:

```
root@crpd-deploy4-674b4fcfb5-8xc5d:~$ sudo kubectl edit deployment/crpd-deploy4
```

```
containers:
 - image: enterprise-hub.juniper.net/crpd-docker-prod/crpd:20.1R1.11
   imagePullPolicy: Never
```

8. Run the following command to confirm if the image is updated:

```
root@crpd-deploy4-674b4fcfb5-8xc5d:~$ sudo kubectl rollout status deployment/crpd-deploy4
```

```
deployment "crpd-deploy4" successfully rolled out
```

9. Run the following command to view the Pods:

```
root@crpd-deploy4-674b4fcfb5-8xc5d> kubectl get pods
```

```
NAME                            READY   STATUS    RESTARTS   AGE
crpd-deploy4-6ff476994d-8z2kr   1/1     Running   0          38s
crpd-deploy4-6ff476994d-qxwrz   1/1     Running   0          41s
```

10. Run the following command to view the image version of the cRPD:

```
root@kubernetes-master:~$ sudo kubectl exec -it crpd-deploy4-6ff476994d-8z2kr -- bash
```

```
===>
           Containerized Routing Protocols Daemon (CRPD)
 Copyright (C) 2018-19, Juniper Networks, Inc. All rights reserved.
                                                                  <===
```

11. Run the following command to view the current image version:

```
root@crpd-deploy4-6ff476994d-8z2kr> show version
```

```
Hostname: crpd-deploy4-674b4fcfb5-8xc5d
Model: cRPD
Junos: 20.4R1.12
cRPD package version : 20.4R1.12 built by builder on 2020-12-20 13:35:15 UTC
```

cRPD Pod Deployment with Allocated Resources

Pods provide two kinds of shared resources namely networking and storage for the containers. When containers in a Pod communicate with entities outside the Pod, they must coordinate how they use the shared network resources (such as ports). Within a Pod, containers communicate through `localhost`using an IP address and port.

Containers within the Pod view the system hostname as same as the configured `name` for the Pod.

Any container in a Pod can enable privileged mode, using the `privileged` flag on the container spec. This is useful for containers that use operating system administrative capabilities such as manipulating the network stack or accessing hardware devices. Processes within a privileged container get almost the same privileges that are available to processes outside a container.

To view the Pod deployment with resources:

1. Create the crpd_res.yaml file on Kubernetes-master add the following text content:

```
------------------
apiVersion: v1
kind: Pod
metadata:
  name: crpdres
  labels:
    app: crpd
spec:
  containers:
  - name: crpd
    image: "enterprise-hub.juniper.net/crpd-docker-prod/crpd:20.1R1.11"
    imagePullPolicy: ""
    ports:
    - containerPort: 179
    - containerPort: 40051
    resources:
      limits:
        memory: "200Mi"
        cpu: "700m"
      requests:
        memory: "200Mi"
        cpu: "700m"
    securityContext:
      privileged: true

------------------
```

2. Save the crpd_res.yaml file to create the crpd Pod.

```
root@kubernetes-master:~# kubectl create -f crpd_res.yaml
```

```
pod/crpdres created
```

3. Run the following command to view the list of existing Pods:

```
root@kubernetes-master:~# kubectl get pods
```

```
NAME                              READY   STATUS    RESTARTS   AGE
crpdres                          1/1     Running            0          7s
```

4. Run the following command to view what containers are inside that Pod and what images are used to build the containers:

```
root@kubernetes-master:~# kubectl describe pod crpres
```

```
Name:               crpdres
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               kubernetes-slave1/10.49.107.224
Start Time:         Thu, 17 Oct 2019 00:28:44 -0700
Labels:             app=crpd
Annotations:        <none>
Status:             Running
IP:                 10.244.1.22
Containers:
  crpd:
    Container ID:   docker://a1ae9791e593b7caea83907d841519bc47744db372b10d006d556308b2e03dbc
    Image:          crpd:19.2R1.8
    Image ID:       docker://
sha256:8e00d0d60309cd0d0bee63fea865b1e389f803a57b1239386e03b31a01146dbf
    Ports:          179/TCP, 40051/TCP
    Host Ports:     0/TCP, 0/TCP
    State:          Running
      Started:      Thu, 17 Oct 2019 00:28:46 -0700
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     700m
      memory:  200Mi
    Requests:
      cpu:        700m
      memory:     200Mi
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-5chxw (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
```

```
  default-token-5chxw:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-5chxw
    Optional:    false
QoS Class:       Guaranteed
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age        From                         Message
  ----    ------     ----       ----                         -------
  Normal  Scheduled  <unknown>  default-scheduler            Successfully assigned default/
crpdres to kubernetes-slave1
  Normal  Pulled     13s        kubelet, kubernetes-slave1   Container image "crpd:19.2R1.8"
already present on machine
  Normal  Created    13s        kubelet, kubernetes-slave1   Created container crpd
  Normal  Started    12s        kubelet, kubernetes-slave1   Started container crpd
```

cRPD Pod Deployment using Mounted Volume

An `emptyDir` is one among the several types of volumes supported on K8s and is first created when a Pod is assigned to a node, and exists as long as that Pod is running on that node. As the name says, the `emptyDir` volume is initially empty. All containers in the Pod can read and write the same files in the `emptyDir` volume, though that volume can be mounted at the same or different paths in each container. When a Pod is removed from a node for any reason, the data in the `emptyDir` is deleted permanently.

To view cRPD Pod deployment by mounting the storage path on Kubernetes:

1. Create the crpd_volume.yaml file on Kubernetes-master add the following text content:

```
------------------
apiVersion: v1
kind: Pod
metadata:
  name: crpd-volume
  labels:
    app: crpd
spec:
  containers:
  - name: crpd
```

```
    image: crpd:19.2R1.8
    imagePullPolicy: Never
    ports:
    - containerPort: 179
    - containerPort: 40051
    volumeMounts:
    - name: crpd-storage
      mountPath: /var/log/crpd-storage
  volumes:
  - name: crpd-storage
    emptyDir: {}
-----------------
```

2. Save the crpd_volume.yaml file to create the crpd Pod.

   root@kubernetes-master:~# **kubectl create -f crpd_volume.yaml**

```
pod/crpd-volume created
```

3. Run the following command to view the list of existing Pods:

   root@kubernetes-master:~# **kubectl get pods**

```
NAME                          READY   STATUS             RESTARTS   AGE
crpd-volume                   1/1     Running            0          5s
```

4. Run the following command to view what containers are inside that Pod and what images are used to build the containers:

   root@kubernetes-master:~# **kubectl describe pod crpd-volume**

```
Name:               crpd-volume
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               kubernetes-slave2/10.49.107.220
Start Time:         Thu, 17 Oct 2019 00:39:59 -0700
Labels:             app=crpd
Annotations:        <none>
Status:             Running
IP:                 10.244.2.20
Containers:
  crpd:
```

```
    Container ID:   docker://593aa6f279132cc2e0a0832cff07ad74db2696472c2d72596a177f1e5f912377
    Image:          crpd:19.2R1.8
    Image ID:       docker://
sha256:82d848c70c24b225fc2ebfa6c39c123153ddde7e3bd5ed40876f537c02693047
    Ports:          179/TCP, 40051/TCP
    Host Ports:     0/TCP, 0/TCP
    State:          Running
      Started:      Thu, 17 Oct 2019 00:40:02 -0700
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/log/crpd-storage from crpd-storage (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-5chxw (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  crpd-storage:
    Type:    EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
  default-token-5chxw:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-5chxw
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age        From                          Message
  ----    ------     ----       ----                          -------
  Normal  Scheduled  <unknown>  default-scheduler             Successfully assigned default/
crpd-volume to kubernetes-slave2
  Normal  Pulled     10s        kubelet, kubernetes-slave2  Container image "crpd:19.2R1.8"
already present on machine
  Normal  Created    9s         kubelet, kubernetes-slave2  Created container crpd
  Normal  Started    9s         kubelet, kubernetes-slave2  Started container crpd
```

5. Run the following command to execute the cRPD instance:

```
root@kubernetes-master:~# kubectl exec -it crpd-volume bash
```

```
===>
           Containerized Routing Protocols Daemon (CRPD)
 Copyright (C) 2018-19, Juniper Networks, Inc. All rights reserved.
                                                                 <===
```

6. Run the following command to view the files in the path:

```
root@crpd-volume:/# ls
```

```
bin  boot  config  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv
sys  tmp  usr  var
```

7. Run the following command to navigate to the storage path:

```
root@crpd-volume:/# cd var/log/crpd-storage/

root@crpd-volume:/var/log/crpd-storage/#
```

Upgrading cRPD

IN THIS SECTION

Upgrade Software

You can upgrade cRPD software by launching a new container using the newer image and attaching the persistent config volumes. Separate volumes are used to store config and logs. They are persistent even after the cRPD is stopped.

NOTE: cRPD does not support in place software upgrade.

To upgrade cRPD:

1. Ensure to import the latest cRPD image. See "Installing cRPD on Docker" on page 20.
2. Load the cRPD software image.

   ```
   root@crpd01:~# docker load -i junos-routing-crpd-docker-20.4R1.12.tgz
   ```
3. Stop the existing container.

   ```
   docker stop crpd01
   ```
4. Run the container using latest version of cRPD.

   ```
   docker run --rm --detach --name crpd01 -h crpd01 --privileged -v crpd01-config:/config -v crpd01-varlog:/var/log -m 2048MB --memory-swap=2048MB -it crpd:20.4R1.12
   ```
5. Enter in to the configuration mode.

   ```
   root@crpd01:~# docker exec -it crpd01 cli
   ```

   ```
   root@crpd01> show version
   ```

```
Hostname: crpd01
Model: cRPD
Junos: 20.4R1.12
cRPD package version : 20.4R1.12 built by builder on 2020-12-20 13:35:15 UTC
```

RELATED DOCUMENTATION

Installing cRPD on Docker | 20

Installing and Configuring cRPD on SONiC

IN THIS SECTION

This section describes on how to load and configure cRPD as a routing stack on Juniper Networks' QFX5210 and QFX5200 switches running Software for Open Networking in the Cloud (SONiC) network operating system.

To install SONiC on QFX5210 and QFX5200 switches, see Install and Upgrade SONiC on Juniper Networks QFX5210 and QFX5200 Switches

How to Load and Start cRPD on SONiC

To load the cRPD package on SONiC, you need to:

- Download the cRPD package from Juniper's software downloads page at https://support.juniper.net/support/downloads/. The cRPD package file name for example is **junos-routing-crpd-docker-20.3R1.8.tgz**

  Transfer the cRPD package to your QFX5210 or QFX5200 switches using scp or sftp file transfer protocol.

The following sections explain how to load and start cRPD on SONiC for QFX5210 or QFX5200 switches:

Load the cRPD Image into Docker

To load the cRPD image into docker, use the `docker load -i` *`junos-routing-crpd-docker-20.3R1.8.tgz`* command as shown below:

```
user@host:~$ docker load -i junos-routing-crpd-docker-20.3R1.8.tgz
b187ff70b2e4: Loading layer [==================================================>] 65.58MB/65.58MB
5930c9e5703f: Loading layer [==================================================>] 991.7kB/991.7kB
c64c52ea2c16: Loading layer [==================================================>] 15.87kB/15.87kB
ddc500d84994: Loading layer [==================================================>] 3.072kB/3.072kB
f76668b91ed5: Loading layer [==================================================>] 40.84MB/40.84MB
cefbbbf6a84d: Loading layer [==================================================>] 7.68kB/7.68kB
19ed2664dd77: Loading layer [==================================================>] 134.5MB/134.5MB
2f362bdab81b: Loading layer [==================================================>] 13.82kB/13.82kB
0d625ccfa452: Loading layer [==================================================>] 3.072kB/3.072kB
75f1d83621fc: Loading layer [==================================================>] 3.584kB/3.584kB
bba7d2bacea8: Loading layer [==================================================>] 3.584kB/3.584kB
911181312301: Loading layer [==================================================>] 3.584kB/3.584kB
98175a8ad5cb: Loading layer [==================================================>] 3.584kB/3.584kB
a113daea3487: Loading layer [==================================================>] 3.584kB/3.584kB
b224ed0cc92d: Loading layer [==================================================>] 3.584kB/3.584kB
48db2eb5713d: Loading layer [==================================================>] 3.584kB/3.584kB
1f620dc1de46: Loading layer [==================================================>] 2.56kB/2.56kB
b9722b673d30: Loading layer [==================================================>] 30.72kB/30.72kB
```

```
cc8250623a79: Loading layer [==================================================>] 6.656kB/6.656kB
5d3819eaf658: Loading layer [==================================================>] 3.584kB/3.584kB
e7ffff983953: Loading layer [==================================================>] 4.096kB/4.096kB
4054102bacd4: Loading layer [==================================================>] 4.096kB/4.096kB
9479c967844e: Loading layer [==================================================>] 4.096kB/4.096kB
91a4575e8d76: Loading layer [==================================================>] 4.096kB/4.096kB
d0aba2dd0145: Loading layer [==================================================>] 4.096kB/4.096kB
25bb582cc7dc: Loading layer [==================================================>] 22.53kB/22.53kB
Loaded image: crpd:20.3R1.8
user@host:~$
```

Verify that cRPD Image is Properly Loaded

To verify if the cRPD image is properly loaded, use the `docker images` command as shown below:

```
user@host:~$ docker images
REPOSITORY                    TAG                         IMAGE ID
CREATED                  SIZE
crpd                          21.2R1.10                   f9b634369718        Less than a
second ago   374MB
docker-fpm-frr                HEAD.0-dirty-20201027.160709   94d35b3d6ff8        Less than a
second ago   335MB
docker-fpm-frr                latest                      94d35b3d6ff8        Less than a
second ago   335MB
docker-syncd-brcm             HEAD.0-dirty-20201027.160709   ef2f75e9156b        Less than a
second ago   436MB
docker-syncd-brcm             latest                      ef2f75e9156b        Less than a
second ago   436MB
docker-router-advertiser      HEAD.0-dirty-20201027.160709   d32efd117a97        Less than a
second ago   289MB
docker-router-advertiser      latest                      d32efd117a97        Less than a
second ago   289MB
docker-sonic-mgmt-framework   HEAD.0-dirty-20201027.160709   b6ebafc68f18        Less than a
second ago   431MB
docker-sonic-mgmt-framework   latest                      b6ebafc68f18        Less than a
second ago   431MB
docker-platform-monitor       HEAD.0-dirty-20201027.160709   ce3c952de93d        Less than a
second ago   357MB
docker-platform-monitor       latest                      ce3c952de93d        Less than a
second ago   357MB
docker-sflow                  HEAD.0-dirty-20201027.160709   05278fdd0019        Less than a
```

```
second ago   315MB
docker-sflow                latest                          05278fdd0019        Less than a
second ago   315MB
docker-lldp-sv2             HEAD.0-dirty-20201027.160709    7f54d84f2da7        Less than a
second ago   312MB
docker-lldp-sv2             latest                          7f54d84f2da7        Less than a
second ago   312MB
docker-dhcp-relay           HEAD.0-dirty-20201027.160709    f86f0bce3b09        Less than a
second ago   299MB
docker-dhcp-relay           latest                          f86f0bce3b09        Less than a
second ago   299MB
docker-database             HEAD.0-dirty-20201027.160709    6daa6a1df857        Less than a
second ago   289MB
docker-database             latest                          6daa6a1df857        Less than a
second ago   289MB
docker-teamd                HEAD.0-dirty-20201027.160709    7596d1a2c302        Less than a
second ago   315MB
docker-teamd                latest                          7596d1a2c302        Less than a
second ago   315MB
docker-snmp-sv2             HEAD.0-dirty-20201027.160709    c258dfe91775        Less than a
second ago   348MB
docker-snmp-sv2             latest                          c258dfe91775        Less than a
second ago   348MB
docker-orchagent            HEAD.0-dirty-20201027.160709    3d602bee0ecb        Less than a
second ago   333MB
docker-orchagent            latest                          3d602bee0ecb        Less than a
second ago   333MB
docker-nat                  HEAD.0-dirty-20201027.160709    0e29ba4560e9        Less than a
second ago   316MB
docker-nat                  latest                          0e29ba4560e9        Less than a
second ago   316MB
docker-sonic-telemetry      HEAD.0-dirty-20201027.160709    521590e31e7d        Less than a
second ago   353MB
docker-sonic-telemetry      latest                          521590e31e7d        Less than a
second ago   353MB
user@host:~$
```

Create and Start the cRPD Container

This section describes how to create, start, and access the cRPD container.

To create the cRPD container, use the `docker create --name crpd -h crpd --net=host --privileged -it crpd:20.3R1.8 b7444647abb7977e0b7eaa884ace8b47bab3632ff2f3f67091d9734a58fa686b` command as shown below:

```
user@host:~$ docker create --name crpd -h crpd --net=host --privileged -it crpd:20.3R1.8
b7444647abb7977e0b7eaa884ace8b47bab3632ff2f3f67091d9734a58fa686b
user@host:~$
```

You need to stop `zebra` and `bgpd` daemons on the BGP container by running the `docker exec bgp supervisorctl stop zebra bgpd` command as shown below:

```
user@host:~$ docker exec bgp supervisorctl stop zebra bgpd
zebra: stopped
bgpd: stopped
user@host:~$
```

To start the cRPD container, use the `docker start crpd` command as shown below:

```
user@host:~$ docker start crpd
crpd
user@host:~$
```

To access the cRPD container, use the `docker exec -it crpd` command as shown below:

```
user@host:~$ docker exec -it crpd cli
root@crpd>
```

Enable RPD connection to fpmsyncd

To enable RPD connection to fpmsyncd, you need to enter the configuration mode and enable `fib-agent` at the [`edit routing-options forwarding-table`] hierarchy level as shown below:

```
root@crpd> configure
Entering configuration mode
[edit]
root@crpd# set routing-options forwarding-table fib-agent
[edit]
root@crpd# commit and-quit
commit complete
```

```
Exiting configuration mode
root@crpd>
```

Once cRPD is up and running, you can configure BGP from the cRPD CLI. The following is a sample BGP configuration:

```
user@host:~$ docker exec -it crpd cli
root@crpd> configure
Entering configuration mode

root@crpd# show protocols
bgp {
    group EBGPv4 {
        type external;
        neighbor 192.168.1.2 {
            description <neighbor_description>;
            local-address 192.168.1.4;
            peer-as 65000;
        }
    }
}
```

CHAPTER 3

Managing cRPD

Syslog Support on cRPD

IN THIS SECTION

Eventd is a process that supports forwarding syslog messages to a configured remote host in containerized RPD (cRPD). You can configure the syslog messages using the following options:

Format	Option	Description
file	*filename*	Eventd writes the syslog messages to the file. You can create a file and forward all the syslog messages to the file based on the priority using the command `set system syslog file <filename> facility priority`.
	`match-strings`	You can filter the messages based on particular string message using the command `set system syslog file test match-strings`.
	`structured data`	You can log the system messages in structured format using the command `set system syslog file test structured data`.
host	*ipaddress*	Host option allows you to log the message in remote host using the command `set system syslog host <ipaddress> <facility> <priority>`.

(Continued)

Format	Option	Description
	match-strings	Match string option with host allows you to filter messages based on particular match string using the command set system syslog host <*ipaddress*> match-strings.
	structured-data	Structured format option at host level allows to log the message to remote host in structured format using the command set system syslog host <*ipaddress*> structured-data.
	log-prefix	Log prefix option at host level allow you to add text string for every syslog message that is forwarded to remote host using the command set system syslog host <*ipaddress*> log-prefix "<*string-name*>".
	source address	Source address option at host level allows you to log the syslog to the remote host with the specified valid source address using the command set system syslog host <*ipaddress*> source address <*ipaddress*> <*facility*> <*priority*>
source address	*ipaddress*	Source address option at syslog level allows you to log the syslog to the remote host with specified source address using the command set system syslog source address <*ipaddress*> file <*file-name*> <*facility*> <*priority*>

Directing System Log Messages to Remote Machine

To direct system log messages to a remote machine, include the `host` statement at the `[edit system syslog]` hierarchy level:

To direct system log messages to a remote machine, include the `host` *`hostname`* statement to specify the remote machine's IPv4 or IPv6 address or fully qualified hostname over WAN port and data port. The remote machine must be running the standard `syslogd` utility. In each system log message directed to the remote machine, the hostname of the local Routing Engine appears after the timestamp to indicate that it is the source for the message.

```
[edit system syslog]
host (hostname) {
    facility severity;
    explicit-priority;
    facility-override facility;
    log-prefix string;
    match "regular-expression";
}
source-address source-address;
```

For the list of logging facilities and severity levels to configure under the `host` statement, see Specifying the Facility and Severity of Messages to Include in the Log.

To record facility and severity level information in each message, include the `explicit-priority` statement. For more information, see Including Priority Information in System Log Messages.

For information about the `match` statement, see Using Strings and Regular Expressions to Refine the Set of Logged Messages.

When directing messages to remote machines, you can include the `source-address` statement to specify the IP address of the switch that is reported in the messages as their source. In each `host` statement, you can also include the `facility-override` statement to assign an alternative facility and the `log-prefix` statement to add a string to each message.

RELATED DOCUMENTATION

- file (System Logging)
- syslog (System)
- Directing System Log Messages to a Log File
- Directing System Log Messages to a User Terminal

Managing cRPD

IN THIS SECTION

Building Topologies

You can use `open-vswitch` to setup topologies and to connect to docker containers. This controls the creation of the bridges, interface naming, and IP addressing.

To build a topology:

1. Download and install `openvswitch-switch` utility.

   ```
   root@ubuntu-vm18:~# apt-get install openvswitch-switch
   ```

2. Navigate to the following path:

   ```
   root@ubuntu-vm18:~# cd /usr/bin
   ```

3. Download ovs-docker with wget:

   ```
   root@ubuntu-vm18:~# wget "https://raw.githubusercontent.com/openvswitch/ovs/master/utilities/ovs-docker"
   ```

4. Change the mode:

   ```
   root@ubuntu-vm18:~# chmod a+rwx ovs-docker
   ```

5. Create the container crpd01.

   ```
   root@ubuntu-vm18:~# docker run --rm --detach --name crpd01 -h crpd01 --net=bridge --privileged -v crpd01-config:/config -v crpd01-varlog:/var/log -it enterprise-hub.juniper.net/crpd-docker-prod/crpd:19.2R1.8
   ```

6. Create the container crpd02.

   ```
   root@ubuntu-vm18:~# docker run --rm --detach --name crpd02 -h crpd02 --net=bridge --privileged -v crpd02-config:/config -v crpd02-varlog:/var/log -it enterprise-hub.juniper.net/crpd-docker-prod/crpd:19.2R1.8
   ```

Networking Docker Containers

Docker containers are connected using user defined bridges. For detailed documentation on docker bridge, see Use Bridge Networks.

To create the docker network:

1. Create a bridge my-net.

   ```
   root@ubuntu-vm18:~# docker network create --internal my-net
   ```

   The `--internal` argument prevents the bridge being connected to the host network which is desirable in some cases. Once a bridge is created, it can be attached or detached to the containers.

2. Connect the two containers using the bridge.

   ```
   root@ubuntu-vm18:~# docker network connect my-net crpd01
   ```

   ```
   root@ubuntu-vm18:~# docker network connect my-net crpd02
   ```

   This creates eth1 with a 172.18.0.0/16 subnet on crpd01 and crpd02.

Removing a Bridge

1. Remove a bridge.

   ```
   root@ubuntu-vm18:~# docker network rm my-net
   ```

2. Disconnect the bridge from the containers.

```
root@ubuntu-vm18:~# docker network disconnect my-net crpd01
root@ubuntu-vm18:~# docker network disconnect my-net crpd02
```

Creating an OVS Bridge

To create an OVS bridge and connect the docker to two containers crpd01 and crpd02:

1. Create a bridge connecting crpd01 and crpd02.

   ```
   root@ubuntu-vm18:~# ovs-vsctl add-br crpd01-crpd02_1
   ```

2. Add interfaces to the bridge.

   ```
   root@ubuntu-vm18:~# ovs-docker add-port crpd01-crpd02_1 eth1 crpd01

   root@ubuntu-vm18:~# ovs-docker add-port crpd01-crpd02_1 eth1 crpd02
   ```

3. Configure an IP address to the interface.

   ```
   root@ubuntu-vm18:~# docker exec -d crpd01 ifconfig eth1 10.1.1.1/24

   root@ubuntu-vm18:~# docker exec -d crpd02 ifconfig eth1 10.1.1.2/24
   ```

4. Configure an IP address to the loopback interface.

   ```
   root@ubuntu-vm18:~# docker exec -d crpd01 ifconfig lo 10.255.255.1 netmask 255.255.255.255

   root@ubuntu-vm18:~# docker exec -d crpd02 ifconfig lo 10.255.255.2 netmask 255.255.255.255
   ```

5. Login to crpd01.

   ```
   root@ubuntu-vm18:~# docker exec -it crpd01 bash
   ```

   ```
   ===>
              Containerized Routing Protocols Daemon (CRPD)
    Copyright (C) 2018-19, Juniper Networks, Inc. All rights reserved.
                                                                    <===
   ```

6. Verify the interface details.

   ```
   root@crpd01:/# ifconfig
   ```

   ```
   eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
           inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
           inet6 fe80::42:acff:fe11:2  prefixlen 64  scopeid 0x20<link>
           ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
           RX packets 28  bytes 2488 (2.4 KB)
           RX errors 0  dropped 0  overruns 0  frame 0
           TX packets 7  bytes 826 (826.0 B)
   ```

```
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.1  netmask 255.255.255.0  broadcast 10.1.1.255
        inet6 fe80::1c2b:50ff:fe9f:6559  prefixlen 64  scopeid 0x20<link>
        ether 1e:2b:50:9f:65:59  txqueuelen 1000  (Ethernet)
        RX packets 364  bytes 33600 (33.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 362  bytes 33748 (33.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 10.255.255.1  netmask 255.255.255.255
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

7. Verify the connection with crpd02

   `root@crpd01:/#` **ping 10.1.1.2 -c 2**

```
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=64 time=0.323 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=64 time=0.042 ms
--- 10.1.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1018ms
rtt min/avg/max/mdev = 0.042/0.182/0.323/0.141 ms
```

Configuring OSPF

1. Configure crpd01 to setup OSPF protocol.

   `root@ubuntu-vm18:~#` **set policy-options policy-statement adv term 1 from route-filter 10.10.10.0/24 exact**

   `root@ubuntu-vm18:~#` **set policy-options policy-statement adv term 1 then accept**

   `root@ubuntu-vm18:~#` **set routing-options router-id 10.255.255.1**

root@ubuntu-vm18:~# **set routing-options static route 10.10.10.0/24 reject**

root@ubuntu-vm18:~# **set protocols ospf export adv**

root@ubuntu-vm18:~# **set protocols ospf area 0.0.0.0 interface eth1**

root@ubuntu-vm18:~# **set protocols ospf area 0.0.0.0 interface lo.0**

2. Configure crpd02 to setup OSPF protocol.

root@ubuntu-vm18:~# **set policy-options policy-statement adv term 1 from route-filter 10.20.20.0/24 exact**

root@ubuntu-vm18:~# **set policy-options policy-statement adv term 1 then accept**

root@ubuntu-vm18:~# **set routing-options router-id 10.255.255.2**

root@ubuntu-vm18:~# **set routing-options static route 10.20.20.0/24 reject**

root@ubuntu-vm18:~# **set protocols ospf export adv**

root@ubuntu-vm18:~# **set protocols ospf area 0.0.0.0 interface eth1**

root@ubuntu-vm18:~# **set protocols ospf area 0.0.0.0 interface lo.0**

3. Login to crpd01.

docker exec -it crpd01 bash

```
===>
           Containerized Routing Protocols Daemon (CRPD)
 Copyright (C) 2018-19, Juniper Networks, Inc. All rights reserved.
                                                                  <===
```

4. Verify OSPF route details.

root@crpd01:/# **cli**

root@crpd01> **show ospf neighbor**

```
Address          Interface              State     ID               Pri  Dead
10.1.1.2         eth1                   Full      10.255.255.2     128    32
```

root@crpd01> **show ospf route**

```
Topology default Route Table:

Prefix             Path  Route      NH       Metric NextHop       Nexthop
```

```
                  Type  Type       Type            Interface     Address/LSP
10.255.255.2      Intra AS BR      IP           1 eth1          10.1.1.2
10.1.1.0/24       Intra Network    IP           1 eth1
10.20.20.0/24     Ext2  Network    IP           0 eth1          10.1.1.2
10.255.255.1/32   Intra Network    IP           0 lo.0
10.255.255.2/32   Intra Network    IP           1 eth1          10.1.1.2
```

root@crpd01> **show route**

```
inet.0: 9 destinations, 9 routes (9 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.1.1.0/24        *[Direct/0] 00:51:32
                    >  via eth1
10.1.1.1/32        *[Local/0] 00:51:32
                       Local via eth1
10.10.10.0/24      *[Static/5] 00:22:21
                       Reject
10.20.20.0/24      *[OSPF/150] 00:20:01, metric 0, tag 0
                    >  to 10.1.1.2 via eth1
10.255.255.1/32    *[Direct/0] 00:25:43
                    >  via lo.0
10.255.255.2/32    *[OSPF/10] 00:20:01, metric 1
                    >  to 10.1.1.2 via eth1
172.17.0.0/16      *[Direct/0] 01:33:53
                    >  via eth0
172.17.0.2/32      *[Local/0] 01:33:53
                       Local via eth0
224.0.0.5/32       *[OSPF/10] 01:33:53, metric 1
                       MultiRecv
...
```

5. Verify the routes.

root@crpd01> **exit**

root@crpd01:/# **ip route**

```
default via 172.17.0.1 dev eth0
10.1.1.0/24 dev eth1 proto kernel scope link src 10.1.1.1
10.20.20.0/24 via 10.1.1.2 dev eth1 proto 22
```

```
10.255.255.2 via 10.1.1.2 dev eth1 proto 22
172.17.0.0/16 dev eth0 proto kernel scope link src 172.17.0.2
```

root@crpd01:/# **ping 10.255.255.2 -c 2**

```
PING 10.255.255.2 (10.255.255.2) 56(84) bytes of data.
64 bytes from 10.255.255.2: icmp_seq=1 ttl=64 time=0.273 ms
64 bytes from 10.255.255.2: icmp_seq=2 ttl=64 time=0.040 ms

--- 10.255.255.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 0.040/0.156/0.273/0.117 ms
```

Removing Interfaces and Bridges

To remove interfaces and bridges:

1. Remove the interfaces:

   root@ubuntu-vm18:~# **ovs-docker del-port crpd01-crpd02_1 eth1 R1**

2. Remove the bridges:

   root@ubuntu-vm18:~# **ovs-vsctl del-br crpd01-crpd02_1**

Viewing Container Processes in a Running cRPD

To view container processes:

Run the `docker exec` command to view the details about the processes (applications, services, and status) running on a container.

root@ubuntu-vm18:~# **docker exec crpd01 ps aux**

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  76996  8060 ?        Ss   Apr26   0:01 /sbin/init
root        19  0.0  0.2 160392 71520 ?        S<s  Apr26   0:38 /lib/systemd/systemd-journald
systemd+    30  0.0  0.0  70616  5236 ?        Ss   Apr26   0:00 /lib/systemd/systemd-resolved
root        32  0.0  0.0 167404 16324 ?        Ssl  Apr26   0:00 /usr/bin/python3 /usr/bin/
networkd-dispatcher --run-startup-triggers
```

```
syslog      33  0.0  0.0 263036  4164 ?        Ssl  Apr26   0:05 /usr/sbin/rsyslogd -n
message+    38  0.0  0.0  49928  4072 ?        Ss   Apr26   0:00 /usr/bin/dbus-daemon --system --
address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
root        47  0.0  0.0  13020  1852 pts/0    Ss+  Apr26   0:00 /sbin/agetty -o -p -- \u --
noclear --keep-baud console 115200,38400,9600 xterm
root        52  0.0  0.0  72296  5536 ?        Ss   Apr26   0:00 /usr/sbin/sshd -D
root        80  0.0  0.0 1453936 13584 ?       Ss   Apr26   0:01 /usr/sbin/mgd -N
root        86  0.1  0.2 1053572 95040 ?       Ssl  Apr26   5:58 /usr/sbin/rpd -N
root        87  0.0  0.0 837400  6356 ?        Ss   Apr26   0:01 /usr/sbin/ppmd -N
root        88  0.0  0.0 842112  6460 ?        Ss   Apr26   0:01 /usr/sbin/bfdd -N
root       102  0.0  0.0  13244  1832 tty1     Ss+  Apr26   0:00 /sbin/agetty -o -p -- \u --
noclear tty1 linux
root       108  0.0  0.0  18500  3340 pts/1    Ss   Apr26   0:00 /bin/bash
root       119  0.0  0.0 739724 11936 pts/1    S+   Apr26   0:02 cli
root       120  0.0  0.0 1454680 12636 ?       Ss   Apr26   0:00 /usr/sbin/mgd -N
root      1502  0.0  0.0  34400  2704 ?        Rs   09:22   0:00 ps aux
```

Accessing cRPD CLI and Bash Shell

To access the cRPD using CLI and bash shell:

1. Run the `docker exec -it crpd1 cli` to launch the Junos CLI.

   `root@ubuntu-vm18:~#` **docker exec -it crpd01 cli**

2. Run the `docker exec -it crpd1 bash` to launch the Junos shell.

   `root@ubuntu-vm18:~#` **docker exec -it crpd01 bash**

Pausing and Resuming Processes within a cRPD Container

You can pause or resume all processes within one or more containers.

To pause and restart a cRPD:

1. Run the `docker pause` command to suspend all the processes in a cRPD container.

   `root@ubuntu-vm18:~#` **docker pause *crpd-container-name***

2. Run the `docker unpause` command to resume all the processes in the cRPD container.

   `root@ubuntu-vm18:~#` **docker unpause *crpd-container-name***

Removing a cRPD Instance

To remove a cRPD instance or image:

> **NOTE**: You must first stop and remove a cRPD instance before you remove a cRPD image.

1. Run the `docker stop` command to stop the cRPD.

   `root@ubuntu-vm18:~#` **docker stop *crpd-container-name***

   ```
   crpd01
   ```

2. Run the `docker rm` command to remove the cRPD.

   `root@ubuntu-vm18:~#` **docker rm *crpd-container-name***

   > **NOTE**: Include `--force` to force the removal of the running cRPD.

3. Run the `docker rmi` command to remove one or more cRPD images from the Docker Engine.

   > **NOTE**: Include `--force` to force the removal a cRPD image.

   `root@ubuntu-vm18:~#` **docker rmi *crd-Image-name***

Viewing Docker Statistics and Logs

To view the statistics and logs:

1. Run the `docker stats` command to monitor the resource utilization.
2. Run the `docker logs` *`crpd-container-name`* command for extracting the container logs.

Viewing Active Containers

To view the current active containers and their status:

Run the `docker ps` or the `docker container ls` command to list the active containers.

root@ubuntu-vm18:~# **docker container ls**

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED       STATUS
PORTS     NAMES
c99e2c74a19b   bad58561c4be           "/storage-provisioner"   12 days ago   Up 12
days             k8s_storage-provisioner_storage-provisioner_kube-system_14f342e7-fa2e-45d1-
a970-6b698f521d3e_11
89c7c630fce2   5fb9aaddb236           "/etc/rc.local init"     3 weeks ago   Up 3
weeks            k8s_csrx_csrx_default_c605afd1-d9ff-4fb7-a290-fc8ce3cad1d7_0
3380dafdb0de   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks            k8s_POD_csrx_default_c605afd1-d9ff-4fb7-a290-fc8ce3cad1d7_0
e779780adc12   bfe3a36ebd25           "/coredns -conf /etc…"   3 weeks ago   Up 3
weeks            k8s_coredns_coredns-f9fd979d6-5nl6b_kube-system_15cfcff1-dbc1-498a-
bf37-02427d30e603_3
7b9506570dec   635b36f4d89f           "/usr/local/bin/kube…"   3 weeks ago   Up 3
weeks            k8s_kube-proxy_kube-proxy-mq9nj_kube-system_841a45cf-de39-49a8-
ae35-6313286c25bb_3
760f482b7cb3   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks            k8s_POD_kube-proxy-mq9nj_kube-system_841a45cf-de39-49a8-ae35-6313286c25bb_3
eb8258e88c9b   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks            k8s_POD_coredns-f9fd979d6-5nl6b_kube-system_15cfcff1-dbc1-498a-
bf37-02427d30e603_3
6d1946fcde75   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks            k8s_POD_storage-provisioner_kube-system_14f342e7-fa2e-45d1-a970-6b698f521d3e_3
8b0842e06094   4830ab618586           "kube-controller-man…"   3 weeks ago   Up 3
weeks            k8s_kube-controller-manager_kube-controller-manager-ix-crpd-01_kube-
system_627d9013c9c4b1cbfb72b4c0ef6cd100_4
bce233248dda   b15c6247777d           "kube-apiserver --ad…"   3 weeks ago   Up 3
weeks            k8s_kube-apiserver_kube-apiserver-ix-crpd-01_kube-
system_a22d3335af147e2c88f1d34b6067e650_7
5f7652e4adda   k8s.gcr.io/etcd        "etcd --advertise-cl…"   3 weeks ago   Up 3
weeks            k8s_etcd_etcd-ix-crpd-01_kube-system_dde4e023d8613808da88a63ff3c86e64_0
8280ab21d826   14cd22f7abe7           "kube-scheduler --au…"   3 weeks ago   Up 3
weeks            k8s_kube-scheduler_kube-scheduler-ix-crpd-01_kube-
system_38744c90661b22e9ae232b0452c54538_3
f451a6be0a98   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks            k8s_POD_etcd-ix-crpd-01_kube-system_dde4e023d8613808da88a63ff3c86e64_0
5c0edfce83be   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks            k8s_POD_kube-scheduler-ix-crpd-01_kube-
system_38744c90661b22e9ae232b0452c54538_0
2d326fedb67c   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks            k8s_POD_kube-controller-manager-ix-crpd-01_kube-
```

```
system_627d9013c9c4b1cbfb72b4c0ef6cd100_0
7e3773affc73   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_kube-apiserver-ix-crpd-01_kube-
system_a22d3335af147e2c88f1d34b6067e650_0
```

root@ubuntu-vm18:~# **docker ps**

```
CONTAINER ID   IMAGE                  COMMAND                  CREATED       STATUS
PORTS     NAMES
c99e2c74a19b   bad58561c4be           "/storage-provisioner"   12 days ago   Up 12
days             k8s_storage-provisioner_storage-provisioner_kube-system_14f342e7-fa2e-45d1-
a970-6b698f521d3e_11
89c7c630fce2   5fb9aaddb236           "/etc/rc.local init"     3 weeks ago   Up 3
weeks             k8s_csrx_csrx_default_c605afd1-d9ff-4fb7-a290-fc8ce3cad1d7_0
3380dafdb0de   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_csrx_default_c605afd1-d9ff-4fb7-a290-fc8ce3cad1d7_0
e779780adc12   bfe3a36ebd25           "/coredns -conf /etc…"   3 weeks ago   Up 3
weeks             k8s_coredns_coredns-f9fd979d6-5nl6b_kube-system_15cfcff1-dbc1-498a-
bf37-02427d30e603_3
7b9506570dec   635b36f4d89f           "/usr/local/bin/kube…"   3 weeks ago   Up 3
weeks             k8s_kube-proxy_kube-proxy-mq9nj_kube-system_841a45cf-de39-49a8-
ae35-6313286c25bb_3
760f482b7cb3   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_kube-proxy-mq9nj_kube-system_841a45cf-de39-49a8-ae35-6313286c25bb_3
eb8258e88c9b   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_coredns-f9fd979d6-5nl6b_kube-system_15cfcff1-dbc1-498a-
bf37-02427d30e603_3
6d1946fcde75   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_storage-provisioner_kube-system_14f342e7-fa2e-45d1-a970-6b698f521d3e_3
8b0842e06094   4830ab618586           "kube-controller-man…"   3 weeks ago   Up 3
weeks             k8s_kube-controller-manager_kube-controller-manager-ix-crpd-01_kube-
system_627d9013c9c4b1cbfb72b4c0ef6cd100_4
bce233248dda   b15c6247777d           "kube-apiserver --ad…"   3 weeks ago   Up 3
weeks             k8s_kube-apiserver_kube-apiserver-ix-crpd-01_kube-
system_a22d3335af147e2c88f1d34b6067e650_7
5f7652e4adda   k8s.gcr.io/etcd        "etcd --advertise-cl…"   3 weeks ago   Up 3
weeks             k8s_etcd_etcd-ix-crpd-01_kube-system_dde4e023d8613808da88a63ff3c86e64_0
8280ab21d826   14cd22f7abe7           "kube-scheduler --au…"   3 weeks ago   Up 3
weeks             k8s_kube-scheduler_kube-scheduler-ix-crpd-01_kube-
system_38744c90661b22e9ae232b0452c54538_3
f451a6be0a98   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_etcd-ix-crpd-01_kube-system_dde4e023d8613808da88a63ff3c86e64_0
```

```
5c0edfce83be   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_kube-scheduler-ix-crpd-01_kube-
system_38744c90661b22e9ae232b0452c54538_0
2d326fedb67c   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_kube-controller-manager-ix-crpd-01_kube-
system_627d9013c9c4b1cbfb72b4c0ef6cd100_0
7e3773affc73   k8s.gcr.io/pause:3.2   "/pause"                 3 weeks ago   Up 3
weeks             k8s_POD_kube-apiserver-ix-crpd-01_kube-
system_a22d3335af147e2c88f1d34b6067e650_0
```

Stopping the Container

To stop the container:

You can stop the container using the following command:

`root@ubuntu-vm18:~#` **docker stop** ***crpd-container-name***

RELATED DOCUMENTATION

Docker commands

Establishing an SSH Connection for a NETCONF Session and cRPD

IN THIS SECTION

Establishing an SSH Connection

SSH can be used to establish connections between a *configuration management server* and a device running Linux OS with cRPD. A configuration management server, as the name implies, is used to configure the device running Linux OS remotely. With SSH, the configuration management server initiates an SSH session with the device running Linux OS.

Enabling SSH

To enable SSH on a cRPD:

1. Log in using the root to enable root access through SSH.
2. In the `/etc/ssh/sshd_config` file, specify the following setting:

   `root@crpd01:/usr/bin#` **vi /etc/ssh/sshd_config**

```
…
# Authentication:

#LoginGraceTime 2m
PermitRootLogin yes
...
```

3. Restart the service.

   `user@crpd01:/#` **service ssh restart**

```
* Starting OpenBSD Secure Shell server
sshd

               [ OK ]
```

Port Forwarding Mechanism

To map a host port to a container port:

Run the following command to map a port on the host with a port on container.

`user@crpd01:/usr/bin#` **docker run -d --name crpd02 -p 8034:22 crpd:20.4R1.12**

Connecting to a NETCONF Server on Container

1. Log in to the container for crpd02.

   root@crpd01:/usr/bin# **docker exec -it crpd02 bash**

2. Copy the IP address.

   root@6918f17c5851:/# **ifconfig eth0**

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.14  netmask 255.255.0.0  broadcast 172.17.255.255
```

3. Configure root-authentication, netconf and ssh, and commit the config

   set system root-authentication plain-text-password "<password>"

   set system services ssh root-login allow

   set system services netconf ssh port 8034

4. Log in to the cRPD container using NETCONF:

   root@crpd01:/usr/bin **ssh root@172.17.0.14 -p 8034 netconf**

```
Password:
<!-- No zombies were killed during the creation of this user interface -->
<!-- user root, class super-user -->
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
...
```

   The 8034 port on host is mapped to 22 port on container and opens an interactive shell session.

4

CHAPTER

Programmable Routing

cRPD Application Development Using JET APIs

IN THIS SECTION

JET is a framework that enables developers to create applications. cRPD supports JET which provides gRPC based routing APIs that are accessed locally from within the container or remotely over a TCP session. JET APIs support Programmable RPD.

JET APIs

cRPD supports the following JET Service APIs:

- **Routing:**
  - PRPD Common API
  - PRPD Service API
  - RIB Service API
  - MPLS Service API
  - Routing Interface Notification API
  - BGP Route Service API

For more information on the APIs, see Juniper Extension Toolkit Developer Guide.

Getting Started with JET

IN THIS SECTION

Configure JET Interaction with Linux OS

In cRPD, JET service is enabled on TCP port 40051 and binds to loopback address 127.0.0.1 or ::1. To access JET service remotely, set up the SSH tunneling into the cRPD container using either username and password or SSH keys.

Remote access to JET service is secured using SSH. For more information on enabling SSH using port forwarding, see "Establishing an SSH Connection for a NETCONF Session and cRPD" on page 76.

The maximum number of JET connections supported is 512.

Maximum Number of JET Connections

To setup maximum number of JET connections:

1. Access the cRPD Linux shell.
2. Use the command to add the connections:

   ```
   root@crpd1:~# ulimit -n 519
   ```
3. Restart ssh on cRPD.

   /etc/init.d/ssh restart
4. Re-establish ssh tunnel from Host Ubuntu to cRPD.
5. Connect up to 512 simultaneous JET Connections.

Compile IDL Files

To download and compile the IDL files:

1. Download the IDL **jet-idl-20.4R1.12.tar.gz** file from the Juniper Networks website at www.juniper.net/support/downloads/.
2. Unpack the IDL file.

   For example:

```
ubuntu-16:~ jet$ mkdir proto
ubuntu-16:~ jet$ tar -xzf jet-idl-18.4-20181107.0.tar.gz -C proto/
ubuntu-16:~ jet$ ls proto/
```

```
1  2 README
```

```
ubuntu-16:~ jet$ ls proto/2
```

```
jnx_authentication_service.proto
jnx_routing_base_service.proto
jnx_common_addr_types.proto
jnx_routing_base_types.proto
jnx_common_base_types.proto
jnx_routing_bgp_service.proto
jnx_firewall_service.proto
jnx_routing_flexible_tunnel_profile.proto
jnx_interfaces_service.proto
jnx_routing_flexible_tunnel_service.proto
jnx_management_service.proto
jnx_routing_interface_service.proto
jnx_registration_service.proto
jnx_routing_rib_service.proto
```

3. Install the **grpcio** module.

   apt-add-repository universe

   apt-get update

apt-get install python-pip

python -m pip install grpcio

python -m pip install grpcio-tools

4. Compile python and gRPC modules for Authentication and Management Service proto files.

   For example:

```
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_management_service.proto
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_authentication_service.proto
ubuntu-16:~ jet$ python -m grpc_tools.protoc -I./proto/2 --python_out=. --grpc_python_out=.
proto/2/jnx_common_base_types.proto
ubuntu-16:~ jet$ ls -lrt
```

```
total 112
-rw-r--r-- 1 vagrant vagrant 52683 Nov  8 16:47 jet-idl-18.4-20181107.0.tar.gz
drwxr-xr-x 1 vagrant vagrant   170 Nov  8 16:49 proto
-rw-r--r-- 1 vagrant vagrant 40924 Nov  8 16:56 jnx_management_service_pb2.py
-rw-r--r-- 1 vagrant vagrant  4719  Nov  8 16:56 jnx_management_service_pb2_grpc.py
-rw-r--r-- 1 vagrant vagrant  5365  Nov 8  2018 jnx_authentication_service_pb2.py
-rw-r--r-- 1 vagrant vagrant  1898  Nov 8  2018 jnx_authentication_service_pb2_grpc.py
-rw-r--r-- 1 vagrant vagrant  6391  Nov 8  2018 jnx_common_base_types_pb2.py
-rw-r--r-- 1 vagrant vagrant    83    Nov 8  2018 jnx_common_base_types_pb2_grpc.py
```

For details on how to generate code from an IDL file in the language of your choice, see https://www.grpc.io/docs.

5

CHAPTER

Using cRPD

Configuring Settings on Host OS

IN THIS SECTION

This chapter provides information on tuning of settings on host OS to enable advanced features or to increase the scale of cRPD functionality.

Configuring ARP Scaling

The maximum ARP entry number is controlled by the Linux host kernel. If there are a large number of neighbors, you might need to adjust the ARP entry limitations on the Linux host. There are options in the `sysctl` command on the Linux host to adjust the ARP or NDP entry limits.

For example, to adjust the maximum ARP entries using IPv4:

```
root@host:~# sysctl -w net.ipv4.neigh.default.gc_thresh1=4096
```

```
root@host:~# sysctl -w net.ipv4.neigh.default.gc_thresh2=8192
root@host:~# sysctl -w net.ipv4.neigh.default.gc_thresh3=8192
```

For example, to adjust the maximum ND entries using IPv6:

```
root@host:~# sysctl -w net.ipv6.neigh.default.gc_thresh1=4096
root@host:~# sysctl -w net.ipv6.neigh.default.gc_thresh2=8192
root@host:~# sysctl -w net.ipv6.neigh.default.gc_thresh3=8192
```

Tunning OSPF under cRPD

To allow more number of OSPFv2/v3 adjacencies with cRPD, increase the IGMP membership limit:

Increase the IGMP membership limit.

```
root@host:~# sysctl -w net.ipv4.igmp_max_memberships=1000
```

Configuring MPLS

To configure MPLS in Linux kernel:

1. Load the MPLS modules in the container using `modprobe` or `insmod` :

   ```
   root@crpd-ubuntu3:~# modprobe mpls_iptunnel
   root@crpd-ubuntu3:~# modprobe mpls_router
   root@crpd-ubuntu3:~# modprobe ip_tunnel
   ```

2. Verify the MPLS modules loaded in host OS.

   ```
   root@host:~# lsmod | grep mpls
   ```

   ```
   mpls_iptunnel          16384  0
   mpls_router            28672  1 mpls_iptunnel
   ip_tunnel              24576  4 ipip,ip_gre,sit,mpls_router
   ```

3. After loading the `mpls_router` on the host, configure the following commands to activate MPLS on the interface.

```
root@host:~# sysctl -w net.mpls.platform_labels=1048575
```

Adding MPLS Routes

Netlink messages are used to communicate (add/learn) the routes with the Linux kernel. MPLS routes are added to the kernel using `iproute2` utility which internally uses netlink socket to update the kernel. To add MPLS routes to host using the `iproute2` utility:

1. Enable mpls on the network interface.

   ```
   root@host:/# cli
   ```

   ```
   root@host> show interfaces routing
   ```

   ```
   Interface     State     Addresses
   eth0.0         Up       MPLS enabled
                           ISO enabled
                           INET 172.17.0.2
                           INET6 fe80::42:acff:fe11:2
   lo.0           Up       MPLS enabled
                           ISO enabled
                           INET 127.0.0.1
                           INET6 ::1
   ```

2. Run the following command to add the mpls routes to the host OS.

   ```
   root@host:~# ip -f mpls route add 100 as 200/300 via inet 172.20.0.2 dev br-a3a2fe3ae8e3
   ```

3. Run the following command to view the MPLS fib entries on Linux.

   ```
   root@host:~# ip -f mpls route show
   ```

   ```
   100 as to 200/300 via inet 172.20.0.2 dev br-a3a2fe3ae8e3
   ```

Adding Routes with MPLS label

To add routes to host by encapsulating the packets with MPLS label using the `iproute2` utility:

1. Run the following command to encapsulate the packets to host OS.

   ```
   root@host:~# ip route add 172.20.0.0/30 encap mpls 200 via inet 172.20.0.2 dev br-a3a2fe3ae8e3
   ```

2. Run the following command to view the mpls routes.

   root@host:~# **ip route show**

```
172.20.0.0/30  encap mpls  200 via 172.20.0.2 dev br-a3a2fe3ae8e3
```

Creating a VRF device

To instantiate a VRF device and associate it with a table:

1. Run the following command to create a VRF device.

   root@host:~# **ip link add dev test1 type vrf table 11**

2. Run the following command to view the created VRFs.

   root@host:~# **ip [-d] link show type vrf**

```
91: test1: <NOARP,MASTER> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether c6:f6:1f:2b:9f:b9 brd ff:ff:ff:ff:ff:ff
```

3. Run the following command to view the list of VRFs in the host OS.

   root@host:~# **ip vrf show**

```
Name              Table
-----------------------
test1               11
```

Assigning a Network Interface to a VRF

Network interfaces are assigned to a VRF by assigning the netdevice to a VRF device. The connected and local routes are automatically moved to the table associated with the VRF device.

To assign a network interface to a VRF:

Run the following command to assign a interface.

root@host:~# **ip link set dev <name> master <name>**

root@host:~# **ip link set dev eth1 vrf test**

Viewing the Devices assigned to VRF

To view the devices:

Run the following command to view the devices assigned to a VRF.

```
root@host:~# ip link show vrf <name>
```

```
root@host:~# ip link show vrf red
```

Viewing Neighbor Entries to VRF

To list the neighbor entries associated with devices enslaved to a VRF device:

Run the following command to add the primary option to the ip command:

```
root@host:~# ip -6 neigh show vrf <NAME>
```

```
root@host:~# ip neigh show vrf red
```

```
root@host:~# ip -6 neigh show vrf red
```

Viewing Addresses for a VRF

To show addresses for interfaces associated with a VRF:

Run the following command to add the primary option to the ip command:

```
root@host:~# ip addr show vrf <NAME>
```

```
root@host:~# ip addr show vrf red
```

Viewing Routes for a VRF

To view routes for a VRF:

1. Run the following command to view the IPv6 routes table associated with the VRF device:

   ```
   root@host:~# ip -6 route show vrf NAME
   ```

   ```
   root@host:~# ip -6 route show table ID
   ```
2. Run the following command to do a route lookup for a VRF device:

root@host:~# **ip -6 route get vrf <NAME> <ADDRESS>**

root@host:~# **ip route get 192.0.2.1 vrf red**

root@host:~# **ip -6 route get oif <NAME> <ADDRESS>**

root@host:~# **ip -6 route get 2001:db8::32 vrf red**

```
2001:db8::32 from :: dev eth1  table red  proto kernel  src 2001:1::2  metric 256  pref medium
```

3. Run the following command to view the IPv4 routes in a VRF device:

   root@host:~# **ip route list table <table-id>**

Removing Network Interface from a VRF

Network interfaces are removed from a VRF by breaking the enslavement to the VRF device

Run the following command to remove the network interface:

root@host:~# **ip link set dev NAME nomaster**

After removing the network interface, connected routes are moved to the default table and local entries are moved to the local table.

Hash Field Selection for ECMP Load Balancing on Linux

You can select the ECMP hash policy (`fib_multipath_hash_policy`) for both forwarded and locally generated traffic (IPv4/IPv6).

IPv4 Traffic

1. By default, Linux kernel uses the Layer 3 hash policy to load balance the IPv4 traffic. Layer 3 hashing uses the following information:
   - Source IP address
   - Destination IP address

   root@host:~# **sysctl -n net.ipv4.fib_multipath_hash_policy 0**
2. Run the following command to load balance the IPv4 traffic using Layer 4 hash policy. Layer 4 hashing load balances the traffic based on the following information:
   - Source IP address

- Destination IP address
- Source port number
- Destination port number
- Protocol

```
root@host:~# sysctl -w net.ipv4.fib_multipath_hash_policy=1

root@host:~# sysctl -n net.ipv4.fib_multipath_hash_policy 1
```

3. Run the following command to use Layer 3 hashing on the inner packet header (IPv4/IPv6 over IPv4 GRE)

```
root@host:~# sysctl -w net.ipv6.fib_multipath_hash_policy=2

root@host:~# sysctl -n net.ipv6.fib_multipath_hash_policy 2
```

The policy defaults to Layer 3 hashing on the packet forwarded as described in the default approach for IPv4 traffic.

IPv6 Traffic

4. By default, Linux kernel uses Layer 3 hash policy to load balance the IPv6 traffic. The Layer 3 hash policy load balance the traffic based on the following information:
   - Source IP address
   - Destination IP address
   - Flow label
   - Next header (Protocol)

```
root@host:~# sysctl -n net.ipv6.fib_multipath_hash_policy 0
```

5. You can use the Layer 4 hash policy to load balance the IPv6 traffic. The Layer 4 hash policy load balances traffic based on the following information:
   - Source IP address
   - Destination IP address
   - Source port number
   - Destination port number
   - Next header (Protocol)

```
root@host:~# sysctl -w net.ipv6.fib_multipath_hash_policy=1

root@host:~# sysctl -n net.ipv6.fib_multipath_hash_policy 1
```

6. Run the following command to use Layer 3 hashing on the inner packet header (IPv4/IPv6 over IPv4 GRE).

   ```
   root@host:~# sysctl -w net.ipv6.fib_multipath_hash_policy=2
   ```

   ```
   root@host:~# sysctl -n net.ipv6.fib_multipath_hash_policy 2
   ```

   MPLS

7. Linux kernel can select the next-hop of a multipath route using the following parameters:

   - label stack upto the limit of **MAX_MP_SELECT_LABELS (4)**
   - source IP address
   - destination IP address
   - protocol of the inner IPv4/IPv6 header

   Neighbor Detection

8. Run the following command to view the liveness (failed/incomplete/unresolved) of the neighbor entry, which helps in forwarding the packets to next-hops.

   ```
   root@host:~# sysctl -w net.ipv4.fib_multipath_use_neigh=1
   ```

   By default, the packets are forwarded to next-hops using the `root@host:~#` **sysctl -n net.ipv4.fib_multipath_use_neigh 0** command.

wECMP using BGP on Linux

Unequal cost load balancing is a way to distribute traffic unequally among different paths (comprising the multipath next-hop); when the paths have different bandwidth capabilities. BGP protocol achieves this by tagging each route/path with the bandwidth of the link using the link bandwidth extended community. The bandwidth of the corresponding link can be encoded as part of this link bandwidth community. RPD uses this bandwidth information of each path to program the multipath next-hops with appropriate linux::weights. A next-hop with linux::weight allows linux kernel to load balance traffic asymmetrically.

BGP forms a multipath next-hop and uses the bandwidth values of individual paths to find out the proportion of traffic that each of the next-hops that form the ECMP next-hop should receive. The bandwidth values specified in the link bandwidth need not be the absolute bandwidth of the interface. These values need to reflect the relative bandwidth of one path from the another. For details, see Understanding How to Define BGP Communities and Extended Communities and How BGP Communities and Extended Communities Are Evaluated in Routing Policy Match Conditions.

Consider a network with R1 receiving equal cost paths from R2 and R3 to a destination R4; if you want to send 90% of the load balanced traffic over the path R1-R2 and the remaining 10% of the traffic over

the path R1-R3 using wECMP, you need to tag routes received from the two BGP peers with link bandwidth community by configuring **policy-options**.

1. Configure policy statement.

   root@host> **show configuration policy-options**

```
policy-statement add-high-bw {
    then {
        community set high-bw;
        accept;
    }
}
policy-statement add-low-bw {
    then {
        community set low-bw;
        accept;
    }
}
community high-bw members [ bandwidth:2:90 ];
community low-bw members [ bandwidth:2:10 ];
```

2. RPD uses the bandwidth values to unequally balance the traffic with the multiple path next-hops.

   root@host> **show route 100.100.100.100 detail**

```
inet.0: 13 destinations, 16 routes (13 active, 0 holddown, 0 hidden)
100.100.100.100/32 (2 entries, 1 announced)
        *BGP    Preference: 170/-101
                Next hop type: Router, Next hop index: 0
                Address: 0x565535f37a3c
                Next-hop reference count: 10
                Source: 10.1.1.5
                Next hop: 20.1.1.5 via eth2 balance 10%, selected
                Session Id: 0x0
                Next hop: 10.1.1.5 via eth1 balance 90%
```

3. Linux kernel supports unequal load balancing by assigning linux::weights for each next-hop.

   root@host:/# **ip route show 100.100.100.100**

```
100.100.100.100 proto 22
```

```
    nexthop via 20.1.1.5 dev eth2 weight 26
    nexthop via 10.1.1.5 dev eth1 weight 229
```

The linux::weights are programmed to linux as divisions of integer 255 (the maximum value of an unsigned character). Each next-hop in the ECMP next-hop is given a linux::weight proportional to its share of the bandwidth.

Enable SRv6 on cRPD

You can enable IPv6 segment routing capability on cRPD using the following sysctl command:

1. To enable segment routing.

   ```
   root@host:~# sysctl net.ipv6.conf.all.seg6_enabled=1

   root@host:~# sysctl net.ipv6.conf.all.forwarding=1
   ```

2. Configure the following command to enable SRv6 on eth0 interface.

   ```
   root@host:~# sysctl net.ipv6.conf.eth0.seg6_enabled=1
   ```

3. Configure the following command to set the DT4 SIDs.

   ```
   root@host:~# sysctl -wq net.vrf.strict_mode=1
   ```

RELATED DOCUMENTATION

Example: Configuring Static Label Switched Paths for MPLS in cRPD | 120

Example: Configuring Layer 3 VPN (VRF) on cRPD Instance | 105

Multitopology Routing in cRPD

IN THIS SECTION

Understanding Multitopology in cRPD

cRPD enables BGP multiple RIBs functionality to support Multitopology routing (MTR) based on the routing policy with Linux FIBs (routes in forwarding plane). The applications can select required routing table based on the routing policy from the Linux FIB in cRPD for different types of traffic. Each type of traffic is defined by a topology that is used to create a new routing table for that topology. Each topology uses the unified control plane to make routing decisions for traffic associated with that topology. In addition, each topology has a separate forwarding table and, in effect, a dedicated forwarding plane for each topology.

Service providers and enterprises can use multitopology routing (MTR) to engineer traffic flow across a network. MTR can be used with direct and static routes, IS-IS, OSPF, and BGP. In a network carrying multiple traffic types, you often need to direct different types of application traffic over multiple links depending on their link characteristics. Communities are used for BGP when exporting routes to multitopology. OSPFv3 does not support MTR. MTR discovers IGP routes and able to resolve BGP routes against the custom topologies with static and OSPF. .

You can configure separate topologies to share the same network links as needed. MTR uses a combination of control plane (routing) and forwarding plane filters.

MTR provides the ability to generate forwarding tables based on the resolved entries in the routing tables for the topologies you create. MTR and forwarding is available only on master routing instance. A dedicated RIB is created for storing the Multitopology routes. BGP multipath is not enabled on topologies.

When routing topologies are configured under `routing-options`, a new routing table for each topology is created. Each routing protocol creates a routing table based on the topology name, the instance name, and the purpose of the table.

Example: Configuring Multitopology Routing with BGP in cRPD

IN THIS SECTION

This example shows how to configure community-based multiple topologies with BGP in cRPD and unicast the traffic using Multitopology Routing (MTR) over network paths.

Requirements

This example requires following software release:

- cRPD 19.4R1 or later.

Overview

IN THIS SECTION

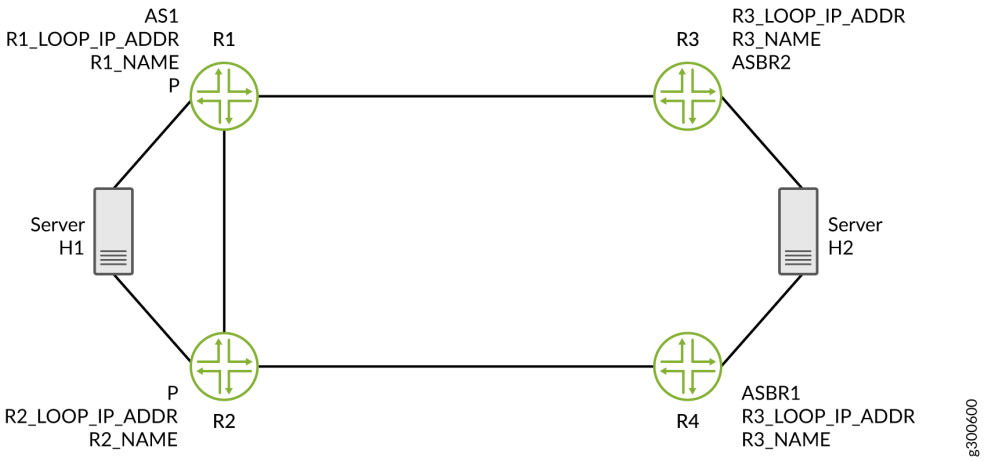
Multitopology routing support for BGP is based on the community value in a BGP route. This configuration determines the association between a topology and one or more community values and populates the topology routing tables. Arriving BGP updates that have a matching community value are replicated in the associated topology routing table.

Configure the topologies with BGP `inet` family and verify the BGP import matching route into topology RIB (also known as routing table). For each topology, a list of community objects must be provided such that the routing software can set up an internal `ribgroup` and the corresponding secondary table import policy.

Topology

Figure 5 on page 97 shows the topology for configuring multitopology routing with BGP.

Figure 5: Multitopology Routing



Configuration

IN THIS SECTION

To configure multitopology routing for BGP:

CLI Quick Configuration

```
set routing-options topologies family inet topology red table-id 40
set routing-options topologies family inet topology blue table-id 41
set routing-options topologies family inet topology green table-id 42
set routing-options router-id 10.2.2.2
set routing-options autonomous-system 65500
set routing-options rib :red.inet.0 static route 10.1.1.1/32 next-hop 10.15.0.2
set routing-options rib :green.inet.0 static route 10.1.1.1/32 next-hop 10.13.0.2
set routing-options rib :blue.inet.0 static route 10.1.1.1/32 next-hop 10.17.0.2
set protocols bgp group ibgp-app-rr-ser type internal
set protocols bgp group ibgp-app-rr-ser traceoptions file bgp size 100m
```

```
set protocols bgp group ibgp-app-rr-ser traceoptions flag update
set protocols bgp group ibgp-app-rr-ser traceoptions flag state
set protocols bgp group ibgp-app-rr-ser local-address 10.77.1.1
set protocols bgp group ibgp-app-rr-ser family inet unicast add-path send path-count 6
set protocols bgp family inet unicast topology red community 1:1
set protocols bgp family inet unicast topology green community 1:2
set protocols bgp family inet unicast topology blue community 1:3
```

Configuring BGP through Multitopology Routing

Step-by-Step Procedure

1. Configure multiple topologies.

```
[edit routing-options topologies]
user@crpd# set family inet topology red table-id 40
user@crpd# set family inet topology blue table-id 41
user@crpd# set family inet topology green table-id 42
```

2. Configure static routes.

```
[edit routing-options]
user@crpd# set router-id 10.2.2.2
user@crpd# set autonomous-system 65500
user@crpd# set rib :red.inet.0 static route 10.1.1.1/32 next-hop 10.15.0.2
user@crpd# set rib :green.inet.0 static route 10.1.1.1/32 next-hop 10.13.0.2
user@crpd# set rib :blue.inet.0 static route 10.1.1.1/32 next-hop 10.17.0.2
```

3. Configure BGP group parameters to import the matching route into the topology routing tables. BGP uses the target community identifier to install the routes it learns in the appropriate routing table.

```
[edit protocols bgp]
user@crpd# set group ibgp-app-rr-ser type internal
user@crpd# set group ibgp-app-rr-ser traceoptions file bgp size 100m
user@crpd# set group ibgp-app-rr-ser traceoptions flag update
user@crpd# set group ibgp-app-rr-ser traceoptions flag state
user@crpd# set group ibgp-app-rr-ser local-address 10.77.1.1
user@crpd# set group ibgp-app-rr-ser family inet unicast add-path send path-count 6
user@crpd# set family inet unicast topology red community 1:1
```

```
user@crpd# set family inet unicast topology green community 1:2
user@crpd# set family inet unicast topology blue community 1:3
```

Results

From configuration mode, confirm your configuration by entering the `show protocols bgp` and `show routing-options` commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```
show routing-options
topologies {
    family inet {
        topology red {
            table-id 40;
        }
        topology blue {
            table-id 41;
        }
        topology green {
            table-id 42;
        }
    }
}
rib :red.inet.0 {
    static {
        route 10.1.1.1/32 next-hop 10.15.0.2;
    }
}
rib :green.inet.0 {
    static {
        route 10.1.1.1/32 next-hop 10.13.0.2;
    }
}
rib :blue.inet.0 {
    static {
        route 10.1.1.1/32 next-hop 10.17.0.2;
```

```
    }
}
```

```
user@crpd# show protocols bgp
family inet {
    unicast {
        topology red {
            community 1:1;
        }
        topology green {
            community 1:2;
        }
        topology blue {
            community 1:3;
        }
    }
    group ibgp-app-rr-ser {
        type internal;
        traceoptions {
            file bgp size 100m;
            flag update;
        }
        local-address 10.77.1.1;
        family inet {
            unicast {
                add-path {
                    send {
                    path-count 6;
                    }
                }
            }
        }
    }
}
```

If you are done configuring the device, enter the `commit` command from configuration mode.

Verification

IN THIS SECTION

Verifying BGP routes

Purpose

To verify the BGP matched routes:

Action

From operational mode, enter the `show route protocol bgp all table` command:

```
user@crpd> show route protocol bgp all table
```

```
:red.inet.0: 11 destinations, 11 routes (8 active, 0 holddown, 3 hidden)
   + = Active Route, - = Last Active, * = Both

   10.99.9.1/32         [BGP/170] 00:05:07, localpref 100, from 10.49.114.118
                         AS path: I, validation-state: unverified
                         >  to 10.15.0.2 via ens4f1

   10.99.9.2/32         [BGP/170] 00:05:07, localpref 100, from 10.49.114.118
                         AS path: I, validation-state: unverified
                          >  to 10.15.0.2 via ens4f1
   10.99.9.5/32         [BGP/170] 00:05:07, localpref 100, from 10.49.114.118
                         AS path: I, validation-state: unverified
                         >  to 10.15.0.2 via ens4f1

   :green.inet.0: 10 destinations, 10 routes (8 active, 0 holddown, 2 hidden)
   + = Active Route, - = Last Active, * = Both

   10.9.9.1/32         [BGP/170] 00:05:07, localpref 100, from 10.49.114.118
                         AS path: I, validation-state: unverified
```

```
                        >  to 10.13.0.2 via ens4f1
   10.9.9.4/32         [BGP/170] 00:05:07, localpref 100, from 10.49.114.118
                         AS path: I, validation-state: unverified
                         >  to 10.13.0.2 via ens4f1

   :blue.inet.0: 11 destinations, 11 routes (8 active, 0 holddown, 3 hidden)
   + = Active Route, - = Last Active, * = Both

   10.99.9.3/32         [BGP/170] 00:05:07, localpref 100, from 10.49.114.118
                         AS path: I, validation-state: unverified
                         >  to 10.17.0.2 via ens4f1
   10.99.9.4/32         [BGP/170] 00:05:07, localpref 100, from 10.49.114.118
                         AS path: I, validation-state: unverified
                         >  to 10.17.0.2 via ens4f1
   10.99.9.5/32         [BGP/170] 00:05:07, localpref 100, from 10.49.114.118
                         AS path: I, validation-state: unverified
                         >  to 10.17.0.2 via ens4f1
```

From operational mode, enter the `show route protocol bgp all table inet.0` command:

```
user@crpd> show route protocol bgp all table inet.0
```

```
inet.0: 20 destinations, 20 routes (20 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.99.9.1/32        *[BGP/170] 00:00:14, localpref 100, from 10.49.114.118
                      AS path: I, validation-state: unverified
                    >  to 1.15.0.2 via ens4f1
10.99.9.2/32        *[BGP/170] 00:00:14, localpref 100, from 10.49.114.118
                      AS path: I, validation-state: unverified
                    >  to 1.15.0.2 via ens4f1
10.99.9.3/32        *[BGP/170] 00:00:14, localpref 100, from 10.49.114.118
                      AS path: I, validation-state: unverified
                    >  to 1.15.0.2 via ens4f1
10.99.9.4/32        *[BGP/170] 00:00:14, localpref 100, from 10.49.114.118
                      AS path: I, validation-state: unverified
                    >  to 1.15.0.2 via ens4f1
10.99.9.5/32        *[BGP/170] 00:00:14, localpref 100, from 10.49.114.118
                      AS path: I, validation-state: unverified
                    >  to 1.15.0.2 via ens4f1
```

Meaning

You can view the BGP matching routes installed to routing tables and when the routes without community targets are available only in the `inet.0` routing table.

SEE ALSO

- Understanding Multitopology Routing
- Understanding Multitopology Routing for Class-Based Forwarding of Voice, Video, and Data Traffic

Layer 3 Overlay Support in cRPD

IN THIS SECTION

Understanding Layer 3 Overlay VRF support in cRPD

IN THIS SECTION

Starting in Junos OS Release 19.4R1, virtual routing and forwarding (VRF) instances are supported in cRPD along with the support of MPLS and Multiprotocol BGP to provide overlay functionality.

A routing instance is a collection of routing tables, interfaces, and routing protocol parameters. To implement Layer 3 VPNs, you configure one routing instance for each VPN. A VRF is a network device in the Linux kernel and the device is associated with `table-id`. You configure the routing instances on PE routers only. You can create VRFs in the Linux network. VRF device implementation impacts only Layer 3 and above. Each VPN routing instance consists of the following components:

- VRF table—On each PE router, you configure one VRF table for each VPN.
- Policy rules—These control the import of routes into and the export of routes from the VRF table.
- One or more routing protocols that install routes from CE routers into the VRF table—You can use the BGP, OSPF, and RIP routing protocols, and you can use static routes.

When a VRF device is created, it is associated with a routing table. Packets that come in through enslaved devices to the VRF are looked up in the routing table associated with the VRF device. Similarly egress routing rules are used to send packets to the VRF driver before sending it out on the actual interface.

VRF is used to manage routes and to forward traffic based on independent forwarding tables in VRF. RPD creates multiple routing tables for every routing instance of type `vrf`. The tables are one for each address family. You need to configure a routing instance for each VPN on each of the PE routers participating in the VPN. You can configure routing instances using the `[edit routing-instances]` hierarchy. The routing instance of type `vrf` is only supported on cRPD.

You can create multiple instances of BFD, BGP, IS-IS, OSPF version 2 (referred as OSPF), OSPF version 3 (OSPFv3), and ICMP router discovery under a VRF using the `[edit routing-instances routing-instance-name protocols]` hierarchy. You can configure protocol independent routing using the `edit routing-instances instance-name routing-options` hierarchy.

Layer-3 Overlay supports the following tunneling protocols in cRPD:

- Static routes in inet.3
- BGP labeled unicast
- GRE tunneling
- MPLS static LSPs
- Routes programmed using programmable-rpd APIs
- direct-ebgp-peering on MPLS enabled interface

Moving the Interfaces under a VRF

The enslavement of devices is done by RPD that is interfaces configured under the routing instance are migrated (enslaved) to the vrf-device by RPD using a netlink message sent to the kernel.

When an interface is configured under the routing instance of type `vrf`, if such a link has been learnt from the kernel and the link is not associated to the correct table, RPD sends a netlink notification to enslave the link. If the link does not exist or RPD has not learnt about the link, whenever the link is created or RPD learns about it then the link will be enslaved correctly based on the configuration.

Example: Configuring Layer 3 VPN (VRF) on cRPD Instance

IN THIS SECTION

This example shows the VPNv4 route resolution on PE routers and route reflectors by configuring the PE routers with specific policies to control the import of routes into and the export of routes from the VRF table and with next hops learnt using BGP labeled unicast. In this example, the traffic flows from CE1 to CE2.

Requirements

This example uses the following hardware and software components:

- Ubuntu software version 18.04
- Linux kernel version 4.5 or later
- cRPD software Release version 19.4R1 or later

Before you configure a Layer 3 VPN (VRF), you must install the basic components:

- MPLS modules on the host OS on which the cRPD instance is created. For details, see "Configuring Settings on Host OS" on page 85.
- Provider edge router (PE1), a provider router (P), and provider edge router (PE2). For installing, see "Installing cRPD on Docker" on page 20.

Overview

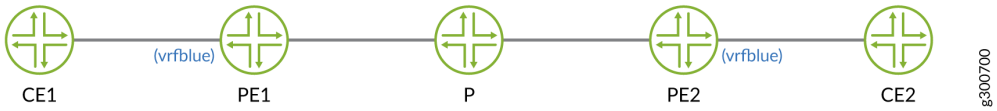
IN THIS SECTION

To configure the VPNv4 route resolution, you need to configure a routing instance of type VRF for each VPN on each of the PE routers participating in the VPN and add static routes to it. The `static` statement configures the static routes that are installed in the `vrfblue.inet.0` routing table. There is no loopback interface or device for every VRF device created in the Linux kernel. But the loopback host addresses are directly added to the VRF device which can be learnt by RPD.

Topology

Figure 6 on page 106 shows the Layer 3 VPN (VRF) Topology

Figure 6: Layer 3 VPN (VRF) Topology



Configuration

IN THIS SECTION

Configuring PE1 router with BGP LU

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy.

1. Create the table mpls.0.

```
user@crpd1# set routing-options rib mpls.0
```

2. Configure policy that accepts routes.

```
[edit policy-options policy-statement]
user@crpd1# set EXPORT_LO term 10 from route-filter 10.2.2.2/32 exact
user@crpd1# set EXPORT_LO term 10 then accept
user@crpd1# set NH_SELF term 10 then next-hop self
```

3. Configure a VRF routing instance on PE1 and other routing instance parameters.

```
[edit routing-instances vrfblue]
user@crpd1# set routing-options static route 10.1.1.1/32 next-hop 10.10.10.1
user@crpd1# set instance-type vrf
user@crpd1# set route-distinguisher 100:100
user@crpd1# set vrf-target target:100:100
```

4. Configure the router ID.

```
user@crpd1# set routing-options router-id 10.2.2.2
```

5. Configure BGP session.

```
[edit protocols bgp group]
user@crpd1# set underlay type external family inet unicast
user@crpd1# set underlay type external export EXPORT_LO neighbor 10.20.20.3 family inet
labeled-unicast resolve-vpn
user@crpd1# set underlay type external export EXPORT_LO neighbor 10.20.20.3 peer-as 65002
local-as 65001
user@crpd1# set VPN type internal local-address 10.2.2.2 family inet-vpn unicast
user@crpd1# set VPN local-as 65005
user@crpd1# set VPN neighbor 10.4.4.4 family inet-vpn unicast
```

6. Configure the interface on MPLS.

```
user@crpd1# set protocols mpls interface all
```

Results

From configuration mode, confirm your configuration by entering the `show protocols bgp` and `show routing-instances` commands. If the output does not display the intended configuration, repeat the configuration instructions in this example to correct it.

```
user@crpd1# show routing-instances
vrfblue {
    routing-options {
        static {
            route 10.1.1.1/32 next-hop 10.10.10.1;
        }
    }
    instance-type vrf;
    route-distinguisher 100:100;
    vrf-target target:100:100;
}
user@crpd1# show protocols bgp
group underlay {
    type external;
    family inet {
        unicast;
    }
    export EXPORT_LO;
    neighbor 10.20.20.3 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
        }
        peer-as 65002;
        local-as 65001;
    }
    neighbor 10.20.20.2 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
        }
        peer-as 65001;
        local-as 65002;
    }
```

```
    neighbor 10.30.30.4 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
        }
        peer-as 65003;
        local-as 65004;
    }
}
group VPN {
    type internal;
    local-address 10.2.2.2;
    family inet-vpn {
        unicast;
    }
    local-as 65005;
    neighbor 10.4.4.4 {
        family inet-vpn {
            unicast;
        }
    }
}
```

If you are done configuring the device, enter commit from configuration mode.

Configuring P router with BGP LU

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy.

1. Create the table mpls.0.

```
user@crpd2# set routing-options rib mpls.0
```

2. Configure policy that accepts routes.

```
[edit policy-options policy-statement]
user@crpd2# set EXPORT_LO term 10 from route-filter 10.3.3.3/32 exact
```

```
user@crpd2# set EXPORT_LO term 10 then accept
user@crpd2# set NH_SELF term 10 then next-hop self
```

3. Configure BGP session.

```
[edit protocols bgp group]
user@crpd2# set underlay type external export EXPORT_LO neighbor 10.20.20.2 family inet
labeled-unicast resolve-vpn
user@crpd2# set underlay type external export EXPORT_LO neighbor 10.20.20.2 peer-as 65001
user@crpd2# set underlay type external export EXPORT_LO neighbor 10.20.20.2 local-as 65002
user@crpd2# set underlay type external export EXPORT_LO neighbor 10.30.30.4 family inet
labeled-unicast resolve-vpn
user@crpd2# set underlay type external export EXPORT_LO neighbor 10.30.30.4 peer-as 65003
user@crpd2# set underlay type external export EXPORT_LO neighbor 10.30.30.4 local-as 65004
```

4. Configure the router ID.

```
user@crpd2# set routing-options router-id 10.3.3.3
```

5. Configure the interface on MPLS.

```
user@crpd2# set protocols mpls interface all
```

Results

From configuration mode, confirm your configuration by entering the `show protocols bgp` and `show policy-options` commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```
user@crpd2# show protocols bgp
group underlay {
    type external;
    export EXPORT_LO;
    neighbor 10.20.20.2 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
```

```
        }
        peer-as 65001;
        local-as 65002;
    }
    neighbor 10.30.30.4 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
        }
        peer-as 65003;
        local-as 65004;
    }
}
```

```
user@crpd2# show policy-options
policy-statement EXPORT_LO {
    term 10 {
        from {
            route-filter 10.3.3.3/32 exact;
        }
        then accept;
    }
}
policy-statement NH_SELF {
    term 10 {
        then {
            next-hop self;
        }
    }
}
```

Configuring PE2 router with BGP LU

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy.

1. Create the table mpls.0.

```
user@crpd3# set routing-options rib mpls.0
```

2. Configure policy that accepts routes.

```
[edit policy-options policy-statement]
user@crpd3# set EXPORT_LO term 10 from route-filter 10.4.4.4/32 exact
user@crpd3# set EXPORT_LO term 10 then accept
user@crpd3# set NH_SELF term 10 then next-hop self
```

3. Configure a VRF routing instance on PE2 and other routing instance parameters.

```
[edit routing-instances vrfblue]
user@crpd3# set routing-options static route 10.5.5.5/32 next-hop 10.40.40.5
user@crpd3# set instance-type vrf
user@crpd3# set route-distinguisher 100:100
user@crpd3# set vrf-target target:100:100
user@crpd3# set interface all
```

4. Configure BGP session.

```
[edit protocols bgp group]
user@crpd3# set underlay type external export EXPORT_LO neighbor 10.30.30.3 family inet
labeled-unicast resolve-vpn
user@crpd3# set underlay type external export EXPORT_LO neighbor 10.30.30.3 peer-as 65004
user@crpd3# set underlay type external export EXPORT_LO neighbor 10.30.30.3 local-as 65003
user@crpd3# set VPN type internal local-address 10.4.4.4 family inet-vpn unicast
user@crpd3# set VPN local-as 65005
user@crpd3# set VPN neighbor 10.2.2.2 family inet-vpn unicast
```

5. Configure the router ID.

```
user@crpd3# set routing-options router-id 10.4.4.4
```

6. Configure the interface on MPLS.

```
user@crpd3# set protocols mpls interface all
```

Results

From configuration mode, confirm your configuration by entering the `show protocols bgp` and `show routing-instances` commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```
user@crpd3# show protocols bgp
group underlay {
    export EXPORT_LO;
    neighbor 10.30.30.3 {
        family inet {
            labeled-unicast {
                resolve-vpn;
            }
        }
        peer-as 65004;
        local-as 65003;
    }
}
group VPN {
    type internal;
    local-address 10.4.4.4;
    family inet-vpn {
        unicast;
    }
    local-as 65005;
    neighbor 10.2.2.2 {
        family inet-vpn {
            unicast;
        }
    }
}
```

```
user@crpd3# show routing-instances
vrfblue {
```

```
    routing-options {
        static {
            route 10.5.5.5/32 next-hop 10.40.40.5;
        }
    }
    interface all;
    instance-type vrf;
    route-distinguisher 100:100;
    vrf-target target:100:100;
}
```

Verification

IN THIS SECTION

Verifying VPNv4 Resolution on PE1

Purpose

To verify VPNv4 routes on PE1:

Action

From operational mode, enter the `show route table vrfblue.inet.0 10.5.5.5` command:

```
user@crpd1> show route table vrfblue.inet.0 10.5.5.5
```

```
vrfblue.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.5.5.5/32         *[BGP/170] 00:00:14, localpref 100, from 10.4.4.4
```

```
                      AS path: I, validation-state: unverified
                    >  to 10.20.20.3 via pe1-p, Push 299808, Push 299792(top)
```

From operational mode, enter the `show route table mpls.0` command:

```
user@crpd1> show route table mpls.0
```

```
mpls.0: 3 destinations, 3 routes (3 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

299808             *[VPN/170] 00:01:45
                    >  to 10.10.10.1 via pe1-ce1, Pop
299808(S=0)        *[VPN/170] 00:01:45
                    >  to 10.10.10.1 via pe1-ce1, Pop
299824             *[VPN/170] 00:01:45
                       receive table vrfblue.inet.0, Pop
```

From bash mode, enter the `ip route list table 5 5.5.5.5` command:

```
user@crpd1> ip route list table 5 10.5.5.5
```

```
10.5.5.5  encap mpls  299792/299808 via 10.20.20.3 dev pe1-p proto 22
```

From bash mode, enter the `ip -f mpls route` command:

```
user@crpd1> ip -f mpls route
```

```
299808 via inet 10.10.10.1 dev pe1-ce1 proto 22
```

Meaning

You can view PE1 has a route under `vrfblue.inet.0` to CE2 which is learnt from PE2 with nexthop 10.4.4.4, which is resolved using BGP LU from P router.

Verifying BGP LU on P

Purpose

To verify VPNv4 routes on P:

Action

From bash mode, enter the `ip -f mpls route show` command:

```
user@crpd2> ip -f mpls route show
```

```
299776 via inet 10.20.20.2 dev p-pe1 proto 22
299792 via inet 10.30.30.4 dev p-pe2 proto 22
```

From operational mode, enter the `show route table mpls.0` command:

```
user@crpd2> show route table mpls.0
```

```
mpls.0: 8 destinations, 8 routes (8 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 01:40:42, metric 1
                       Receive
1                  *[MPLS/0] 01:40:42, metric 1
                       Receive
2                  *[MPLS/0] 01:40:42, metric 1
                       Receive
13                 *[MPLS/0] 01:40:42, metric 1
                       Receive
299776             *[VPN/170] 01:19:24
                    >  to 10.20.20.2 via p-pe1, Pop
299776(S=0)        *[ VPN/170] 01:19:24
                    >  to 10.20.20.2 via p-pe1, Pop
299792             *[VPN/170] 01:19:20
                    >  to 10.30.30.4 via p-pe2, Pop
299792(S=0)        *[VPN/170] 01:19:20
                    >  to 10.30.30.4 via p-pe2, Pop
```

Meaning

You can view the MPLS and VPN routes from P to PE1 and P to PE2.

Verifying VPNv4 Resolution on PE2

Purpose

To verify VPNv4 routes on PE2:

Action

From operational mode, enter the `show route table vrfblue.inet.0 10.1.1.1` command:

```
user@crpd3> show route table vrfblue.inet.0 10.1.1.1
```

```
vrfblue.inet.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.1.1.1/32         *[BGP/170] 00:00:26, localpref 100, from 10.2.2.2
                      AS path: I, validation-state: unverified
                    >  to 10.30.30.3 via pe2-p, Push 299808, Push 299776(top)
```

From operational mode, enter the `show route table mpls.0` command:

```
user@crpd3> show route table mpls.0
```

```
mpls.0: 7 destinations, 7 routes (7 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 01:34:39, metric 1
                       Receive
1                  *[MPLS/0] 01:34:39, metric 1
                       Receive
2                  *[MPLS/0] 01:34:39, metric 1
                       Receive
13                 *[MPLS/0] 01:34:39, metric 1
                       Receive
```

```
299808             *[VPN/170] 00:00:43
                    >  to 10.40.40.5 via pe2-ce2, Pop
299808(S=0)        *[VPN/170] 00:00:43
                    >  to 10.40.40.5 via pe2-ce2, Pop
299824             *[VPN/170] 00:00:43
                       receive table vrfblue.inet.0, Pop
```

From bash mode, enter the `ip route list table 5 10.1.1.1` command:

```
user@crpd3> ip route list table 5 10.1.1.1
```

```
10.1.1.1  encap mpls  299776/299808 via 10.30.30.3 dev pe2-p proto 22
```

From bash mode, enter the `ip -f mpls route` command:

```
user@crpd3> ip -f mpls route
```

```
299808 via inet 10.40.40.5 dev pe2-ce2 proto 22
```

Meaning

On PE2 router, PE1 displays the routes for the VRF table `vrfblue.inet.0` using BGP LU about 10.1.1.1 as a VPNv4 prefix with nexthop as 10.2.2.2.

SEE ALSO

- vrf-target
- vrf-import
- route-distinguisher
- vrf-export
- vrf-table-label
- no-vrf-advertise
- Routing Instances in Layer 3 VPNs

MPLS Support in cRPD

IN THIS SECTION

Understanding MPLS support in cRPD

Multiprotocol Label Switching (MPLS) configuration is supported in cRPD for forwarding packets to the destination in MPLS network.

With MPLS, only the first device does a routing lookup. Instead of finding the next hop, the device finds the ultimate destination along with a path to that destination. The path of an MPLS packet is called a label-switched path (LSP). LSPs are unidirectional routes through a network or an autonomous system (AS). MPLS routers within an AS determine paths through a network through the exchange of MPLS traffic engineering information. Using these paths, the routers direct traffic through the network along an established route. Rather than selecting the next hop along the path as in IP routing, each router is responsible for forwarding the packet to a predetermined next hop address.

Routers that are part of the LSP are label-switching routers (LSRs). An MPLS LSP is established using static LSPs. A static LSP requires each router along the path to be configured explicitly. You must manually configure the path and its associated label values.

cRPD supports only a limited number of Junos OS MPLS features. You can configure MPLS `interface`, `ipv6-tunneling`, `label-history`, `label-range`, and `static-label-switched-path` in cRPD CLI under the `edit protocols mpls` hierarchy.

Supported Features

- BGP configuration
- MPLS using PRPD API
- BGP labeled unicast configuration

SEE ALSO

mpls

static-label-switched-path

Example: Configuring Static Label Switched Paths for MPLS in cRPD

Configuring Settings on Host OS | 85

Example: Configuring Static Label Switched Paths for MPLS in cRPD

IN THIS SECTION

This example shows how the VPN traffic flows through a v4 MPLS tunnel among PEs by configuring BGP and MPLS static label switched paths.

Requirements

This example uses the following hardware and software components:

- Ubuntu software version 18.04
- Linux kernel version 4.5 or later
- cRPD software Release version 19.4R1 or later

Before you configure a static LSP for MPLS forwarding, you must install the basic components:

- MPLS modules on host OS on which cRPD instance is created. For details, see "Configuring Settings on Host OS" on page 85.
- Provider edge router (PE1), a provider router (P), and provider edge router (PE2). For installing, see "Installing cRPD on Docker" on page 20.

Overview

IN THIS SECTION

In this example, PE1 acts as a Label Edge Router or ingress node to the MPLS network, which encapsulates the packets by attaching labels. P acts as Label Switching Router that transfers MPLS packets using labels in the MPLS network.

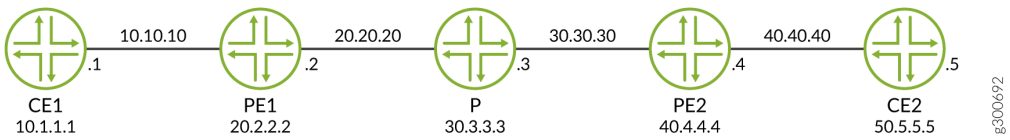
To configure MPLS, you must first create one or more named paths on the ingress and transit routers. For each path, you can specify some or all transit routers in the path.

Configuring static label-switched paths (LSPs) for MPLS is similar to configuring static routes on individual routers.

Topology

Figure 7 on page 121 shows the topology used in this example.

Figure 7: MPLS Forwarding in cRPD



Configuration

IN THIS SECTION

To configure static LSP for MPLS on cRPD:

Configuring PE1 Router

Step-by-Step Procedure

To configure the static LSP:

1. Create the tables inet.0 and mpls.0.

```
[edit routing-options]
user@crpd1# set rib inet.0
user@crpd1# set rib mpls.0
user@crpd1# set router-id 20.2.2.2
```

2. Configure BGP session.

```
[edit protocols bgp group VPN]
user@crpd1# set type internal local-address 20.2.2.2 family inet-vpn unicast
user@crpd1# set local-as 5
user@crpd1# set neighbor 40.4.4.4 family inet-vpn unicast
```

3. Configure the static label range and ingress static LSP parameters.

```
[edit protocols mpls]
user@crpd1# set interface all
user@crpd1# set label-range static-label-range 1000000 1048575
user@crpd1# set static-label-switched-path pe2 ingress install 40.4.4.4/32 active
user@crpd1# set static-label-switched-path pe2 ingress to 40.4.4.4 next-hop 20.20.20.2 push
1000001
```

4. Configure a static route from the ingress PE2.

```
[edit routing-options static]
user@crpd1# set route 20.2.2.2/32 next-hop 20.20.20.2
user@crpd1# set route 40.4.4.4/32 static-lsp-next-hop pe2
```

5. Configure a VRF routing instance on PE1 and other routing instance parameters.

```
[edit routing-instances vrfblue]
user@crpd1# set routing-options static route 10.1.1.1/32 next-hop 10.10.10.1
user@crpd1# set route-distinguisher 100:100
user@crpd1# set vrf-target target:100:100
user@crpd1# set interface all
```

Results

From configuration mode, confirm your configuration by entering the `show protocols bgp` and `run show configuration protocols mpls` commands on PE1. If the output does not display the intended configuration, repeat the configuration instructions in this example to correct it.

```
user@crpd1# show protocols bgp
group VPN {
    type internal;
    local-address 20.2.2.2;
    family inet-vpn {
        unicast;
    }
    local-as 5;
    neighbor 40.4.4.4 {
        family inet-vpn {
            unicast;
        }
    }
}
```

```
user@crpd1# run show configuration protocols mpls
interface all;
static-label-switched-path pe2 {
    ingress {
        next-hop 20.20.20.3;
        to 40.4.4.4;
        push 1000001;
    }
}
```

If you are done configuring the device, enter commit from configuration mode.

Configuring Provider P Router.

Step-by-Step Procedure

To configure the static LSP:

1. Configure router ID for router P.

```
[edit routing-options]
user@crpd2# set rib mpls.0
user@crpd2# set router-id 30.3.3.3
```

2. Configure a transit static LSP for swap and pop labels.

```
[edit protocols mpls]
user@crpd2# set label-range static-label-range 1000000 1048575
user@crpd2# set static-label-switched-path pe2 transit 1000001 next-hop 30.30.30.4 swap
1000002
user@crpd2# set static-label-switched-path pe1 transit 1000003 next-hop 20.20.20.2 swap
1000004
user@crpd2# set static-label-switched-path pe2 transit 1000001 pop next-hop 30.30.30.4
user@crpd2# set static-label-switched-path pe1 transit 1000003 pop next-hop 20.20.20.2
```

Results

From configuration mode, confirm your configuration by entering the `show protocols bgp`, `run show configuration protocols mpls`, and `run show mpls interface` commands on P. If the output does not display the intended configuration, repeat the configuration instructions in this example to correct it.

```
user@crpd2# run show configuration protocols mpls
interface all;
static-label-switched-path pe1 {
    transit 1000003 {
        next-hop 20.20.20.2;
        swap 1000004;
    }
}
static-label-switched-path pe2 {
```

```
    transit 1000001 {
        next-hop 30.30.30.4;
        swap 1000002;
    }
}
```

If you are done configuring the device, enter commit from configuration mode.

Configuring PE2 Router

Step-by-Step Procedure

To configure the static LSP for MPLS on PE2:

1. Configure BGP session.

```
[edit protocols bgp group VPN ]
user@crpd3# set type internal local-address 40.4.4.4 family inet-vpn unicast
user@crpd3# set local-as 5
user@crpd3# set neighbor 20.2.2.2 family inet-vpn unicast
```

2. Configure the ingress static LSP parameters.

```
[edit protocols mpls ]
user@crpd3# set interface all
user@crpd3# set label-range static-label-range 1000000 1048575
user@crpd3# set static-label-switched-path pe1 ingress install 20.2.2.2/32 active
user@crpd3# set static-label-switched-path pe1 ingress to 20.2.2.2 next-hop 30.30.30.4 push
1000003
```

3. Configure router ID and a static route from the ingress PE1.

```
[edit routing-options]
user@crpd3# set rib inet.0
user@crpd3# set router-id 40.4.4.4
user@crpd3# set static route 40.4.4.4/32 next-hop 30.30.30.4
user@crpd3# set static route 20.2.2.2/32 static-lsp-next-hop pe1
```

4. Configure a VRF routing instance on PE2 and other routing instance parameters.

```
[edit routing-instances vrfblue]
user@crpd3# set routing-options static route 50.5.5.5/32 next-hop 40.40.40.5
user@crpd3# set route-distinguisher 100:100
user@crpd3# set vrf-target target:100:100
user@crpd3# set interface all
```

Results

From configuration mode, confirm your configuration by entering the `run show configuration protocols mpls` and `run show mpls interface` commands on PE2. If the output does not display the intended configuration, repeat the configuration instructions in this example to correct it.

```
user@crpd3# show protocols bgp
group VPN {
    type internal;
    local-address 40.4.4.4;
    family inet-vpn {
        unicast;
    }
    local-as 5;
    neighbor 20.2.2.2 {
        family inet-vpn {
            unicast;
        }
    }
}
```

```
user@crpd3# run show configuration protocols mpls
interface all;
static-label-switched-path pe2 {
    ingress {
        next-hop 20.20.20.3;
        to 40.4.4.4;
        push 1000001;
    }
}
```

If you are done configuring the device, enter commit from configuration mode.

Verification

IN THIS SECTION

Verify MPLS forwarding on PE1

Purpose

To verify the configuration for MPLS on PE1.

Action

From operational mode, enter the `show route table vrfblue.inet.0 50.5.5.5` command:

```
user@crpd1> show route table vrfblue.inet.0 50.5.5.5
```

```
vrfblue.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

50.5.5.5/32         *[BGP/170] 00:01:03, localpref 100, from 40.4.4.4
                      AS path: I, validation-state: unverified
                    >  to 20.20.20.3 via pe1-p, Push 299776, Push 1000001(top)
```

From operational mode, enter the `show mpls label usage` command:

```
user@crpd1> show mpls label usage
```

```
Label space Total   Available        Applications
LSI         999984  999983 (100.00%) BGP/LDP VPLS with no-tunnel-services, BGP L3VPN with vrf-
```

```
table-label
Block       999984  999983 (100.00%) BGP/LDP VPLS with tunnel-services, BGP L2VPN
Dynamic     999984  999983 (100.00%) RSVP, LDP, PW, L3VPN, RSVP-P2MP, LDP-P2MP, MVPN, EVPN, BGP
Static      48576   48576  (100.00%) Static LSP, Static PW
Effective Ranges
Range name  Shared with Start   End
Dynamic     16      999999
Static      1000000 1048575
Configured Ranges
Range name  Shared with Start   End
Dynamic     16      999999
Static      1000000 1048575
```

From operational mode, enter the `show mpls static-lsp` command:

```
user@crpd1> show mpls static-lsp
```

```
Ingress LSPs:
LSPname                             To              State
pe2                                 40.4.4.4         Up
Total 1, displayed 1, Up 1, Down 0

Transit LSPs:
Total 0, displayed 0, Up 0, Down 0

Bypass LSPs:
Total 0, displayed 0, Up 0, Down 0

Segment LSPs:
Total 0, displayed 0, Up 0, Down 0
```

From operational mode, enter the `show route table inet.3` command:

```
user@crpd1> show route table inet.3
```

```
inet.3: 1 destinations, 1 routes (1 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
40.4.4.4/32         *[MPLS/6/1] 00:04:44, metric 0
                    >  to 20.20.20.3 via pe1-p, Push 1000001
```

From operational mode, enter the `show route table mpls.0` command:

```
user@crpd1> show route table mpls.0
```

```
mpls.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 00:15:45, metric 1
                       Receive
1                  *[MPLS/0] 00:15:45, metric 1
                       Receive
2                  *[MPLS/0] 00:15:45, metric 1
                       Receive
13                 *[MPLS/0] 00:15:45, metric 1
                       Receive
299776             *[VPN/170] 00:06:32
                    >  to 10.10.10.1 via pe1-ce1, Pop
299776(S=0)        *[VPN/170] 00:06:32
                    >  to 10.10.10.1 via pe1-ce1, Pop
```

From operational mode, enter the `ip route list table 5 50.5.5.5` command:

```
user@crpd1> ip route list table 5 50.5.5.5
```

```
50.5.5.5  encap mpls  1000001/299776 via 20.20.20.3 dev pe1-p proto 22
```

From operational mode, enter the `ip -f mpls route` command:

```
user@crpd1> ip -f mpls route
```

```
299776 via inet 10.10.10.1 dev pe1-ce1 proto 22
```

Verify MPLS forwarding on P

Purpose

To verify the configuration for MPLS on P.

Action

From shell mode, enter the `show route table mpls.0` command:

```
user@crpd2> show route table mpls.0
```

```
mpls.0: 10 destinations, 10 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

0                  *[MPLS/0] 00:00:11, metric 1
                       Receive
1                  *[MPLS/0] 00:00:11, metric 1
                       Receive
2                  *[MPLS/0] 00:00:11, metric 1
                       Receive
13                 *[MPLS/0] 00:00:11, metric 1
                       Receive
299776             *[VPN/170] 00:00:05
                    >  to 20.20.20.2 via p-pe1, Pop
299776(S=0)        *[VPN/170] 00:00:05
                    >  to 20.20.20.2 via p-pe1, Pop
299792             *[VPN/170] 00:00:05
                    >  to 30.30.30.4 via p-pe2, Pop
299792(S=0)        *[VPN/170] 00:00:05
                    >  to 30.30.30.4 via p-pe2, Pop
1000001            *[MPLS/6] 00:00:11, metric 1
                    >  to 30.30.30.4 via p-pe2, Swap 1000002
```

```
1000003            *[MPLS/6] 00:00:11, metric 1
                    >  to 20.20.20.2 via p-pe1, Swap 1000004
```

```
user@crpd2> show mpls static-lsp
```

```
Ingress LSPs:
Total 0, displayed 0, Up 0, Down 0

Transit LSPs:
LSPname                              Incoming-label  State
pe1                                  1000003         Up
pe2                                  1000001         Up
Total 2, displayed 2, Up 2, Down 0

Bypass LSPs:
Total 0, displayed 0, Up 0, Down 0

Segment LSPs:
Total 0, displayed 0, Up 0, Down 0
```

From bash shell mode, enter the `ip -f mpls route` command:

```
user@crpd2:/# ip -f mpls route
```

```
299776 via inet 20.20.20.2 dev p-pe1 proto 22
299792 via inet 30.30.30.4 dev p-pe2 proto 22
1000001 as to 1000002 via inet 30.30.30.4 dev p-pe2 proto 22
1000003 as to 1000004 via inet 20.20.20.2 dev p-pe1 proto 22
```

Verify MPLS forwarding on PE2

Purpose

To verify the configuration for MPLS on P.

Action

From shell mode, enter the `show route table vrfblue.inet.0 10.1.1.1` command:

```
user@crpd3> show route table vrfblue.inet.0 10.1.1.1
```

```
vrfblue.inet.0: 5 destinations, 5 routes (5 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.1.1.1/32         *[BGP/170] 00:03:00, localpref 100, from 2.2.2.2
                      AS path: I, validation-state: unverified
                    >  to 30.30.30.3 via pe2-p, Push 299776, Push 1000003(top)
```

```
user@crpd3> show mpls static-lsp
```

```
Ingress LSPs:
LSPname                             To              State
pe1                                 20.2.2.2         Up
Total 1, displayed 1, Up 1, Down 0

Transit LSPs:
LSPname                             Incoming-label  State
pe2                                 1000002         Dn
Total 1, displayed 1, Up 0, Down 1

Bypass LSPs:
Total 0, displayed 0, Up 0, Down 0

Segment LSPs:
Total 0, displayed 0, Up 0, Down 0
```

```
user@crpd3> show route table mpls.0
```

```
mpls.0: 6 destinations, 6 routes (6 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both
```

```
0                  *[MPLS/0] 00:17:31, metric 1
                      Receive
1                  *[MPLS/0] 00:17:31, metric 1
                      Receive
2                  *[MPLS/0] 00:17:31, metric 1
                      Receive
13                 *[MPLS/0] 00:17:31, metric 1
                      Receive
299776             *[VPN/170] 00:03:07
                    >  to 40.40.40.5 via pe2-ce2, Pop
299776(S=0)        *[VPN/170] 00:03:07
                    >  to 40.40.40.5 via pe2-ce2, Pop
```

From bash shell mode, enter the `ip -f mpls route` command:

```
user@crpd3:/# ip -f mpls route
```

```
299776 via inet 40.40.40.5 dev pe2-ce2 proto 22
```

From bash shell mode, enter the `ip route list table 5 10.1.1.1` command:

```
user@crpd3:/# ip route list table 5 10.1.1.1
```

```
10.1.1.1  encap mpls  1000003/299776 via 30.30.30.3 dev pe2-p proto 22
```

Meaning

You can verify the static LSP between PEs are up on all the devices and the routes are populated in the corresponding route tables `inet.0` and `inet.3` and in the Linux FIB.

SEE ALSO

- mpls
- static-label-switched-path

Sharding and UpdateIO on cRPD

IN THIS SECTION

Understanding Sharding

The BGP process is split into different threads so that they can run concurrently on a multicore routing engine through RIB sharding which results in reduced convergence time and faster performance. BGP RIB sharding splits a BGP RIB into several sub RIBs and each sub RIB handles a subset of BGP routes. Each sub RIB is served by a separate RPD thread to achieve parallel processing.

BGP RIB sharding is disabled by default. This feature is supported only on 64-bit routing protocol process (rpd) where the Routing Engine has more than one core. We recommend configuring this feature on a device with atleast 4CPU cores and 16GB of memory.

If you configure rib-sharding on a routing engine, RPD will create sharding threads. By default the number of sharding threads created is same as the number of CPU cores on the routing engine. Optionally, you can specify the number-of-shards you want to create. The range is currently 1 through 31.

BGP RIB sharding on crpd is supported for `inet.0` and `inet6.0` as well as `bgp.l3vpn.0`, `bgp.l3vpn-inet6.0` and `bgp.rtarget`, `inet-vpn`, `inet6-vpn unicast`, and `route-target` tables.

To enable this feature, you can configure `rib-sharding` at the `edit system processes routing bgp` hierarchy level. Sharding is dependent on the UpdateIO thread feature. Therefore, UpdateIO thread feature is mandatory when you configure sharding. To enable updateIO, you need to configure `update-threading` at the `[edit system processes routing bgp]` hierarchy level for `rib-sharding` configuration to pass commit check.

SEE ALSO

- rib-sharding
- show bgp neighbor
- show route summary

show bgp summary

show route

Understanding UpdateIO

The BGP protocol work to do Update message generation for peers in a BGP group. The BGP work in main RPD thread is split into different threads, called BGP UpdateIO threads. Each UpdateIO thread is responsible for generating updates for one or more BGP peer groups. BGP Update threads construct updates for groups in parallel and independent of other groups that are being serviced by different update threads.

This might offer significant convergence improvement in a write-heavy workload that involves advertising to many peers spread across many groups. BGP UpdateIO threads can be configured independent of RIB sharding feature but are mandatory to use with RIB sharding as they help improve packing of prefixes in outbound BGP update messages and thus help improve performance.

BGP update thread is disabled by default. If you configure update-threading on a routing engine, RPD creates update threads. By default, the number of update threads created is the same as the number of CPU cores on the routing engine. Update threading is only supported on a 64 bit routing protocol process (RPD). Optionally, you can specify the number-of-threads you want to create by using `set update-threading <number-of-threads>` statement at the `edit system processes routing bgp` hierarchy level. The range is currently 1 through 128.

SEE ALSO

Understanding BGP Update IO Thread

update-threading

VRRP with cRPD

IN THIS SECTION

Overview

IN THIS SECTION

The Virtual Router Redundancy Protocol (VRRP) in cRPD eliminates the single point of failure in the static default route environment. VRRP dynamically assigns responisbility for a virtual router to one of the VRRP routers. The VRRP router that controls the IP address associated with the virtual router is called Master. Master forwards the packets to these IP address.

When the master router is unavailable, any of the virtual routers IP address is used as the default first hop router by end hosts.

Benefit of VRRP

Supports high availability default path without configuring dynamic routing on every end host.

How VRRP Works with cRPD?

SUMMARY

This example configuration provides steps on how to configure active/back up load balancing feature using VRRP in cRPD. To achieve this support, you must configure VRRP instance with a virtual address in two cRPD containers. One as a master and other as a backup. When the failover happens the virtual address is taken by the backup router from the master router.

1. Configure VRRP on cRPD instance 1. The virtual address is used as a next-hop for the route.

```
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111
virtual-address 10.0.0.254/32 device-name ens3f1
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111
```

```
priority 200
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111
advertise-interval 1
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 mode
master
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111
authentication-type simple
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111
authentication-key 12345
```

2. Configure VRRP with track and notify script. Notify script is to trigger script execution when the VRRP state changes to master/backup. Track script is used to monitor every state transition of instance.

```
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 track
interface ens3f2 weight cost 10
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 track
track-script keepalived_check script-name /etc/crpd/scripts/track_script_sample.sh
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 track
track-script keepalived_check interval 10
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 track
track-script keepalived_check time-out 5
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 track
track-script keepalived_check rise 3
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 track
track-script keepalived_check fall 3
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 track
track-script keepalived_check weight cost 10
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.1/24 vrrp-group 111 track
notify-script /etc/crpd/scripts/keepalived_notify.sh
```

3. Configure VRRP on cRPD instance 2.

```
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111
virtual-address 10.0.0.254/32 device-name ens3f1
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111
priority 200
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111
advertise-interval 1
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 mode
```

```
backup
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111
authentication-type simple
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111
authentication-key 12345
```

4. Configure VRRP with track and notify script.

```
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 track
interface ens3f2 weight cost 10
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 track
track-script keepalived_check script-name /etc/crpd/scripts/track_script_sample.sh
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 track
track-script keepalived_check interval 10
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 track
track-script keepalived_check time-out 5
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 track
track-script keepalived_check rise 3
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 track
track-script keepalived_check fall 3
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 track
track-script keepalived_check weight cost 10
user@crpd1# set interfaces ens3f1 unit 0 family inet address 10.0.0.2/24 vrrp-group 111 track
notify-script /etc/crpd/scripts/keepalived_notify.sh
```

Use the following commands to verify that cRPD1 has transitioned to backup state and that the router is backup for group ens3f1_v4_111.

```
user@crpd1> show vrrp
Interface  State   Group     VR-state   VR Mode  Interface-address   Virtual-address
ens3f1   up        ens3f1_v4_111 backup   Inactive  10.0.0.1/24            10.0.0.254/32
```

```
user@crpd1> show vrrp track
Track Int     State       VRRP int       Group  VR State Priority
ens3f2        up          ens3f1  ens3f1_v4_111   backup    200
```

Use the following commands to verify that the instance has transitioned to master state and that the router is master for group ens3f1_v4_111.

```
user@crpd1> show vrrp
Interface  State   Group     VR-state   VR Mode  Interface-address   Virtual-address
ens3f1   up        ens3f1_v4_111 master   Active    10.0.0.2/24             10.0.0.254/32
```

```
user@crpd1> show vrrp track
Track Int     State       VRRP int       Group  VR State Priority
ens3f2        up          ens3f1  ens3f1_v4_111   master    200
```

RELATED DOCUMENTATION

No Link Title

No Link Title

CHAPTER 6

Troubleshooting

Debugging cRPD Application

IN THIS SECTION

Troubleshooting is a systematic approach to solving a problem. The goal of troubleshooting is to determine why something does not work as expected and how to resolve the problem.

Command-Line Interface

The Junos OS command-line interface (CLI) is the primary tool for controlling and troubleshooting router hardware, the Junos OS, routing protocols, and network connectivity. CLI commands display information from routing tables, information specific to routing protocols, and information about network connectivity derived from the `traceroute` utilities. RPD tracelog facilities are supported and enabled through the CLI. Trace log files are stored `/var/log` path.

You can use the following Junos CLI commands to troubleshoot cRPD:

- `show task`: Display the routing protocol tasks on the Routing Engine.
- `show task memory` *`detail`*: Display the memory utilization for routing protocol tasks on the Routing Engine.
- `show route`: Display the active entries in the routing tables.
- `show bfd`: Display information about active Bidirectional Forwarding Detection (BFD) sessions.
- `show bgp`: Display information about BGP summary information for all routing instances.

- `show (ospf | ospf3)`: Display standard information about all OSPF neighbors for all routing instances.
- `show interfaces routing`: Perform router diagnostics.
- `show log`: View system activity logs and allows you to monitor and view information for performance monitoring, troubleshooting, and debugging purposes.
- `show krt`: Monitor KRT queues and their states.
- `show programmable-rpd`: List clients connected to the programmable routing protocol process (prpd) server. The prpd provides public APIs to program routing systems, making it possible for users to directly access the APIs to customize, create, and modify the behavior of their network.
- `ip monitor`: Monitor the installation of routes to Linux FIB and interface events and netlink messages.
- `tcpdump`: Capture network traffic to/from control plane.
- `netstat`: Monitor the sockets.
- `request support information`: Display the support information which is used for troubleshooting.

Fault Handling

When the rpd crashes due to some issue, the rpd process is restarted automatically. To recover manually from a fault, you can implement the following CLI command hierarchies to handle the faults:

- `restart routing`: Restart the rpd.
- `clear bgp`: Clear BGP sessions.
- `deactivate`: Deactivate CLI configuration.
- `activate`: Activate the CLI configuration.

Troubleshooting Container

You can implement various docker commands to monitor and troubleshoot issues at container level when cRPD is deployed as a docker container.

- `docker ps`: List out active containers and their state.
- `docker stats`: Continuous monitor the resource utilization.

- `docker logs`: Extract container logs in case the container terminates unexpectedly.
- `docker stop`: Stop the Docker from the current state.
- `docker start`: Restart the Docker container.

Verify Docker

1. Verify the installed Docker Engine version by using the `docker version` command.

   `root@ubuntu-vm18:~#` **docker version**

```
Client:
     Version:           18.09.1
     API version:       1.39
     Go version:        go1.10.6
     Git commit:        4c52b90
     Built:             Wed Jan  9 19:35:31 2019
     OS/Arch:           linux/amd64
     Experimental:      false

    Server: Docker Engine - Community
     Engine:
      Version:          18.09.1
      API version:      1.39 (minimum version 1.12)
      Go version:       go1.10.6
      Git commit:       4c52b90
      Built:            Wed Jan  9 19:02:44 2019
      OS/Arch:          linux/amd64
      Experimental:     false
```

2. View the software and hardware information in the system.

   `root@ubuntu-vm18:~#` **uname -a**

```
Linux ubuntu-vm18 4.15.0-43-generic #46-Ubuntu SMP Thu Dec 6 14:45:28 UTC 2018 x86_64 x86_64
x86_64 GNU/Linux
```

3. View the version of ubuntu.

```
root@ubuntu-vm18:~# lsb_release -a

No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:        18.04
Codename:       bionic
```

Viewing Core Files

IN THIS SECTION

Purpose

When a core file is generated, you can find the output at **/var/crash**. The core files generated are stored on the system that is hosting the Docker containers.

You can also use ping and ping6 to check the reachability at the shell mode.

Action

To list the core files:

1. Exit from the CLI environment to return to the host unix shell.

   ```
   user@host> start shell
   ```

2. Change the directory to **/var/crash**:

   ```
   root@ubuntu-vm18$ cd /var/crash
   ```

   ```
   root@ubuntu-vm18$ ls -l
   ```

3. Run the command to identify the location of the core files:

```
root@ubuntu-vm18$ sysctl kernel.core_pattern
```

4. Verify for any core files created around the time of the crash.

Configuring Syslog

Syslog is enabled by default and the messages are stored at **/var/log/messages** file stored on the local Routing Engine.

To configure remote syslog:

1. Access the cRPD Linux shell.
2. Open the **/etc/rsyslog.conf** file.
3. Add the following facility information:

   ***.* @<IP address>:<port>**

   Where: <IP address> is the IP address of the remote syslog server.
4. Save the file.
5. Restart syslog by using the following command:

   ```
   root@crpd1# service rsyslog restart
   ```

To view the log messages:

1. You can view the log messages using the following command:

   ```
   root@crpd1> show log messages
   ```

SEE ALSO

Log File Sample Content

Display Plain Text Version of Obfuscated (9) or Encrypted (8) Password

You can use the following command to show plain text versions of obfuscated (9) or encrypted (8) passwords present in configuration files:

```
root@crpd1> request system decrypt password
```

For more information, see request system decrypt password.

Troubleshooting with Kubectl

IN THIS SECTION

Troubleshooting is a systematic approach to solving a problem. The goal of troubleshooting is to determine why something does not work as expected and how to resolve the problem.

Kubectl Command-Line Interface

You can use the following Kubectl commands to troubleshoot cRPD:

- `kubectl get`: Lists the resources.
- `kubectl describe` : Display detailed information about a resource.
- `kubectl logs`: Display the logs from a container in a pod.
- `kubectl exec`: Execute a command on a container in a pod.

Viewing Pods

A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker or rkt), and some shared resources for those containers.

Those resources include:

- Shared storage, as Volumes
- Networking, as a unique cluster IP address
- Information about how to run each container, such as the container image version or specific ports to use

When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster. Summary: Pods Nodes Kubectl main commands A Pod is a group of one or more application containers (such as Docker or rkt) and includes shared storage (volumes), IP address and information about how to run them.

Run the following command to view pods:

```
root@ubuntu-vm18:~# kubectl get pods
```

Viewing Container Logs

IN THIS SECTION

Purpose

Anything that the application sends to STDOUT becomes logs for the container within the Pod.

Action

To view the logs:

1. Run the following command to retrieve the logs:

   ```
   root@ubuntu-vm18$ kubectl logs $POD_NAME
   ```

Debugging EVPN VXLAN on RPD and Linux

SUMMARY

IN THIS SECTION

Before you start debugging for EVPN over VXLAN support in cRPD, ensure you have the configuration created.

Configuring EVPN Over VXLAN

Configure Layer 2 EVPN over VXLAN with MAC-VRF on cRPD .

```
routing-instances {
    evpn-vxlan {
        instance-type mac-vrf;
        protocols {
            evpn {
                encapsulation vxlan;
                default-gateway do-not-advertise;
            }
        }
        service-type vlan-aware;
        vtep-source-interface lo.0;
        bridge-domains {
            bd600 {
                vlan-id 600;
                interface ens3f2.600;
                routing-interface irb.600;
                vxlan {
                    vni 2600;
                    destination-udp-port 4790;
```

```
                }
            }
            bd601 {
                vlan-id 601;
                interface ens3f3.601;
                routing-interface irb.601;
                vxlan {
                    vni 2601;
                    destination-udp-port 4790;
                }
            }
        }
        route-distinguisher 81.1.1.1:1;
        vrf-target target:1:1;
    }
}
interfaces {
    irb {
        unit 600 {
            family inet {
                address 99.60.0.254/24;
            }
            family inet6 {
                address 1234::99.60.0.254/120;
            }
        }
        unit 601 {
            family inet {
                address 99.60.1.254/24;
            }
            family inet6 {
                address 1234::99.60.1.254/120;
            }
        }
    }
}
```

Verifying Layer 2 EVPN Over VXLAN Support in cRPD

1. Verify the bridge device is created in RPD and Linux kernel.

root@PE1_CRPD> **show evpn instance evpn-vxlan extensive**

RPD view

```
Instance: evpn-vxlan
  Route Distinguisher: 81.1.1.1:1
  Encapsulation type: VXLAN
  Control word enabled
  Duplicate MAC detection threshold: 5
  Duplicate MAC detection window: 180
  MAC database status                       Local  Remote
    MAC advertisements:                         3       2
    MAC+IP advertisements:                      9       6
    Default gateway MAC advertisements:         2       0
  Number of local interfaces: 3 (3 up)
    Interface name  ESI                            Mode             Status     AC-Role
    .local..2       00:00:00:00:00:00:00:00:00:00  single-homed     Up         Root
    ens3f2.600      00:00:00:00:00:00:00:00:00:00  single-homed     Up         Root
    ens3f3.601      00:00:00:00:00:00:00:00:00:00  single-homed     Up         Root
  Number of IRB interfaces: 2 (2 up)
    Interface name  VLAN   VNI    Status  L3 context
    irb.600                2600    Up     evpn-vrf
    irb.601                2601    Up     evpn-vrf
  Number of protect interfaces: 0
  Number of bridge domains: 2
    VLAN  Domain-ID Intfs/up   IRB-intf  Mode            MAC-sync IM-label  MAC-label  v4-SG-
sync IM-core-NH v6-SG-sync IM-core-NH Trans-ID
    600   2600         1  1    irb.600   Extended        Enabled  2600
Disabled             Disabled             2600
    601   2601         1  1    irb.601   Extended        Enabled  2601
Disabled             Disabled             2601
  Number of neighbors: 1
    Address               MAC    MAC+IP        AD        IM        ES Leaf-label Remote-DCI-
Peer
    81.2.2.2               2         6         0         2         0
  Number of ethernet segments: 2
    ESI: 05:00:00:00:7b:00:00:0a:28:00
      Local interface: irb.600, Status: Up/Forwarding
    ESI: 05:00:00:00:7b:00:00:0a:29:00
      Local interface: irb.601, Status: Up/Forwarding
  Router-ID: 81.1.1.1
```

```
  Source VTEP interface IP: 81.1.1.1
  SMET Forwarding: Disabled
```

root@PE1_CRPD> **show krt table | grep evpn-vxlan**

```
evpn-vxlan.evpn-mac.0         : GF: 11 krt-index: 7    ID: 0 kernel-id: 2
```

Kernel view

root@PE1_CRPD:/# **ip link show __crpd-brd2**

__crpd-brd<2> is kernel id from **show krt table**

```
148: __crpd-brd2: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group
default qlen 1000
   link/ether 56:68:a3:1a:07:9c brd ff:ff:ff:ff:ff:ff
   alias evpn-vxlan
```

root@PE1_CRPD:/# **ip -d link show __crpd-brd2**

```
148: __crpd-brd2: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group
default qlen 1000
           link/ether 56:68:a3:1a:07:9c brd ff:ff:ff:ff:ff:ff promiscuity 0
           bridge forward_delay 1500 hello_time 200 max_age 2000 ageing_time 30000 stp_state
0 priority 32768 vlan_filtering 1 vlan_protocol 802.1Q bridge_id 8000.56:68:a3:1a:7:9c
designated_root
8000.56:68:a3:1a:7:9c root_port 0 root_path_cost 0 topology_change 0 topology_change_detected
0 hello_timer    0.00 tcn_timer    0.00 topology_change_timer    0.00 gc_timer   54.32
vlan_default_pvid 0
vlan_stats_enabled 0 group_fwd_mask 0 group_address 01:80:c2:00:00:00 mcast_snooping 0
mcast_router 1 mcast_query_use_ifaddr 0 mcast_querier 0 mcast_hash_elasticity 4
mcast_hash_max 512
mcast_last_member_count 2 mcast_startup_query_count 2 mcast_last_member_interval 100
mcast_membership_interval 26000 mcast_querier_interval 25500 mcast_query_interval 12500
mcast_query_response_interval
1000 mcast_startup_query_interval 3124 mcast_stats_enabled 0 mcast_igmp_version 2
mcast_mld_version 1 nf_call_iptables 0 nf_call_ip6tables 0 nf_call_arptables 0 addrgenmode
eui64 numtxqueues 1
```

```
numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
           alias evpn-vxlan
```

2. Verify if the VXLAN devices are created corresponding to the VXLAN configuration under bridge domains.

RPD view

VXLAN configs of interest under routing-instance bridge-domains.

```
routing-instances {
    evpn-vxlan {
        bridge-domains {
            bd600 {
                ...
                vxlan {
                    vni 2600;
                    destination-udp-port 4790;
                }
            }
            bd601 {
                ...
                vxlan {
                    vni 2601;
                    destination-udp-port 4790;
                }
            }
        }
    }
}
```

Kernel view

`root@PE1_CRPD:/#` **ip -d link show vxlan2600**

```
16: vxlan2600: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-brd2 state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 0e:6b:fd:27:a5:63 brd ff:ff:ff:ff:ff:ff promiscuity 1
    vxlan id 2600 local 81.1.1.1 srcport 0 0 dstport 4790 nolearning tos inherit ttl 100
ageing 300 noudpcsum noudp6zerocsumtx noudp6zerocsumrx
    bridge_slave state forwarding priority 32 cost 100 hairpin off guard off root_block off
```

```
fastleave off learning off flood
on port_id 0x8003 port_no 0x3 designated_port 32771 designated_cost 0 designated_bridge
8000.e:6b:fd:27:a5:63
designated_root 8000.e:6b:fd:27:a5:63 hold_timer    0.00 message_age_timer    0.00
forward_delay_timer
0.00 topology_change_ack 0 config_pending 0 proxy_arp off proxy_arp_wifi off mcast_router 1
mcast_fast_leave off mcast_flood on
neigh_suppress on group_fwd_mask 0x0 group_fwd_mask_str 0x0 vlan_tunnel off addrgenmode eui64
numtxqueues 1 numrxqueues 1
gso_max_size 65536 gso_max_segs 65535
```

root@PE1_CRPD:/# **ip -d link show vxlan2601**

```
17: vxlan2601: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-brd2 state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 32:82:1d:c2:e9:8b brd ff:ff:ff:ff:ff:ff promiscuity 1
    vxlan id 2601 local 81.1.1.1 srcportdstport 4790  0 0 nolearning tos inherit ttl 100
ageing 300 noudpcsum noudp6zerocsumtx noudp6zerocsumrx
    bridge_slave state forwarding priority 32 cost 100 hairpin off guard off root_block off
fastleave off learning off flood on port_id 0x8004 port_no
0x4 designated_port 32772 designated_cost 0 designated_bridge 8000.e:6b:fd:27:a5:63
designated_root 8000.e:6b:fd:27:a5:63 hold_timer
0.00 message_age_timer    0.00 forward_delay_timer    0.00 topology_change_ack 0
config_pending 0 proxy_arp off proxy_arp_wifi off
mcast_router 1 mcast_fast_leave off mcast_flood on neigh_suppress on group_fwd_mask 0x0
group_fwd_mask_str 0x0 vlan_tunnel off addrgenmode eui64
numtxqueues 1 numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
```

3. Verify all the instance interfaces (bridge domain interfaces including vxlan devices) are enslaved to bridge device in kernel.

   RPD view

   Interface configs of interest under routing-instance bridge-domains.

```
routing-instances {
    evpn-vxlan {
        ...
        bridge-domains {
            bd600 {
                ...
                interface ens3f2.600;
```

```
                vxlan {
                    vni 2600;   -> vxlan2600
                }
            }
            bd601 {
                ...
                interface ens3f3.601;
                vxlan {
                    vni 2601;   -> vxlan2601
                }
            }
        }
    }
}
```

Kernel view

Ensure all the instance IFL have "master __crpd-brd2" which means they are ensalved to __crpd-brd2 bridge device through ip link.

`root@PE1_CRPD:/#` **ip link show master __crpd-brd2**

```
12: ens3f2.600@ens3f2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-
brd2 state UP mode DEFAULT group default qlen 1000
    link/ether 56:68:a3:54:20:b7 brd ff:ff:ff:ff:ff:ff
13: ens3f3.601@ens3f3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-
brd2 state UP mode DEFAULT group default qlen 1000
    link/ether 56:68:a3:54:20:bb brd ff:ff:ff:ff:ff:ff
16: vxlan2600: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-brd2 state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 0e:6b:fd:27:a5:63 brd ff:ff:ff:ff:ff:ff
17: vxlan2601: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-brd2 state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 32:82:1d:c2:e9:8b brd ff:ff:ff:ff:ff:ff
19: irbbe-brd2@irbve-brd2: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-brd2
state UP mode DEFAULT group default qlen 1000
    link/ether fe:72:e9:b0:b5:92 brd ff:ff:ff:ff:ff:ff
```

4. Verify if all the instance interfaces which are part of the bridge device are assigned to vids matching the bridge-domain on RPD.

   RPD view

VLAN/interface configs of interest under routing-instance bridge-domains.

```
routing-instances {
    evpn-vxlan {
        ...
        bridge-domains {
            bd600 {
                vlan-id 600; --->bd600/vid
                interface ens3f2.600;
                vxlan {
                    vni 2600;   -> vxlan2600
                }
            }
            bd601 {
                vlan-id 601; --->bd601/vid
                interface ens3f3.601;
                vxlan {
                    vni 2601;   -> vxlan2601
                }
            }
        }
    }
}
```

Kernel view

`root@PE1_CRPD:/#` **bridge vlan show**

```
port    vlan ids
ens3f2.600       600 PVID Egress Untagged
ens3f3.601       601 PVID Egress Untagged
__crpd-brd2      None
vxlan2600        600 PVID Egress Untagged
vxlan2601        601 PVID Egress Untagged
irbbe-brd2       600
                 601
```

5. Verify if irb interface (vlan subinterface with bridge-domains vlan-id) is created in kernel corresponding to the routing-interface configuration under bridge-domains.

   RPD view

IRB interface configs of interest under routing-instance bridge-domains.

```
routing-instances {
    evpn-vxlan {
        ...
        bridge-domains {
           bd600 {
                vlan-id 600;
                routing-interface irb.600;
            }
            bd601 {
                vlan-id 601;
                routing-interface irb.601;
            }
        }
    }
}
```

Kernel view

`root@PE1_CRPD:/#` **ip -d link show irb.600**

```
20: irb.600@irbve-brd2: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-vrf1
state UP mode DEFAULT group default qlen 1000
    link/ether d6:a3:f9:94:70:78 brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 600 <REORDER_HDR>
    vrf_slave table 1 addrgenmode eui64 numtxqueues 1 numrxqueues 1 gso_max_size 65536
gso_max_segs 65535
```

`root@PE1_CRPD:/#` **ip -d link show irb.601**

```
22: irb.601@irbve-brd2: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-vrf1
state UP mode DEFAULT group default qlen 1000
    link/ether d6:a3:f9:94:70:78 brd ff:ff:ff:ff:ff:ff promiscuity 0
    vlan protocol 802.1Q id 601 <REORDER_HDR>
    vrf_slave table 1 addrgenmode eui64 numtxqueues 1 numrxqueues 1 gso_max_size 65536
gso_max_segs 65535
```

6. Ensure if ipv4/ipv6 addresses are assigned to the irb interfaces.

RPD view

IP address configs of IRB interfaces.

```
interfaces {
    irb {
        unit 600 {
            family inet {
                address 99.60.0.254/24;
            }
            family inet6 {
                address 1234::99.60.0.254/120;
            }
        }
        unit 601 {
            family inet {
                address 99.60.1.254/24;
            }
            family inet6 {
                address 1234::99.60.1.254/120;
            }
        }
    }
}
```

Kernel view

`root@PE1_CRPD:/#` **ip addr show irb.600**

```
20: irb.600@irbve-brd2: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-vrf1
state UP group default qlen 1000
    link/ether d6:a3:f9:94:70:78 brd ff:ff:ff:ff:ff:ff
    inet 99.60.0.254/24 scope global irb.600
       valid_lft forever preferred_lft forever
    inet6 1234::633c:fe/120 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::d4a3:f9ff:fe94:7078/64 scope link
       valid_lft forever preferred_lft forever
```

root@PE1_CRPD:/# **ip addr show irb.601**

```
22: irb.601@irbve-brd2: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master __crpd-vrf1
state UP group default qlen 1000
    link/ether d6:a3:f9:94:70:78 brd ff:ff:ff:ff:ff:ff
    inet 99.60.1.254/24 scope global irb.601
       valid_lft forever preferred_lft forever
    inet6 1234::633c:1fe/120 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::d4a3:f9ff:fe94:7078/64 scope link
       valid_lft forever preferred_lft forever
```

7. Verify bridge flood entries are created in kernel, correpsonding to the received IM (inclusive multicast) route entries received from peers.

   RPD view

   root@PE1_CRPD> **show route table evpn-vxlan.evpn.0 protocol bgp | grep IM**

```
3:81.2.2.2:1::2600::81.2.2.2/248 IM
3:81.2.2.2:1::2601::81.2.2.2/248 IM
```

   Kernel view

   root@PE1_CRPD:/# **bridge fdb show br __crpd-brd2 state static | grep 00:00:00:00:00:00**

```
00:00:00:00:00:00 dev vxlan2600 dst 81.2.2.2 self static
00:00:00:00:00:00 dev vxlan2601 dst 81.2.2.2 self static
```

8. Verify local MAC entries are learnt and advertised by EVPN to remote peers.

   RPD view

   root@PE1_CRPD> **show evpn database instance evpn-vxlan origin local**

```
Instance: evpn-vxlan
VLAN  DomainId  MAC address        Active source                  Timestamp        IP address
     2600       00:11:11:11:60:00  ens3f2.600                     May 10 23:49:46  99.60.0.1
```

```
1234::633c:1

fe80::5668:a302:5854:1f14
     2600       d6:a3:f9:94:70:78  irb.600                       Apr 29 21:08:59  99.60.0.254

1234::633c:fe

fe80::d4a3:f9ff:fe94:7078
     2601       00:11:11:11:60:10  ens3f3.601                    May 10 23:47:44  99.60.1.1

1234::633c:101

fe80::5668:a302:5954:1f15
     2601       d6:a3:f9:94:70:78  irb.601                       Apr 29 21:08:59  99.60.1.254

1234::633c:1fe

fe80::d4a3:f9ff:fe94:7078
```

root@PE1_CRPD> **show route table evpn-vxlan.evpn.0 protocol evpn | grep MAC**

```
2:81.1.1.1:1::2600::00:11:11:11:60:00/304 MAC/IP
2:81.1.1.1:1::2601::00:11:11:11:60:10/304 MAC/IP
2:81.1.1.1:1::2600::00:11:11:11:60:00::99.60.0.1/304 MAC/IP
2:81.1.1.1:1::2601::00:11:11:11:60:10::99.60.1.1/304 MAC/IP
2:81.1.1.1:1::2600::00:11:11:11:60:00::1234::633c:1/304 MAC/IP
2:81.1.1.1:1::2600::00:11:11:11:60:00::fe80::5668:a302:5854:1f14/304 MAC/IP
2:81.1.1.1:1::2601::00:11:11:11:60:10::1234::633c:101/304 MAC/IP
2:81.1.1.1:1::2601::00:11:11:11:60:10::fe80::5668:a302:5954:1f15/304 MAC/IP
```

Kernel view

Mac entries are learnt from **bridge fdb** table

root@PE1_CRPD:/# **bridge fdb show br __crpd-brd2 brport ens3f2.600 state dynamic**

```
00:11:11:11:60:00 vlan 600 master __crpd-brd2
```

```
root@PE1_CRPD:/# bridge fdb show br __crpd-brd2 brport ens3f3.601 state dynamic

00:11:11:11:60:10 vlan 601 master __crpd-brd2
```

Mac+ip bindings are learnt from **ip neigh** table

```
root@PE1_CRPD:/# ip neigh show dev irb.600 | grep -v PERMANENT

99.60.0.1 lladdr 00:11:11:11:60:00 REACHABLE
1234::633c:1 lladdr 00:11:11:11:60:00 router STALE
fe80::5668:a302:5854:1f14 lladdr 00:11:11:11:60:00 router STALE
```

```
root@PE1_CRPD:/# ip neigh show dev irb.601 | grep -v PERMANENT

99.60.1.1 lladdr 00:11:11:11:60:10 REACHABLE
1234::633c:101 lladdr 00:11:11:11:60:10 router STALE
fe80::5668:a302:5954:1f15 lladdr 00:11:11:11:60:10 router STALE
```

9. Verify remote MAC entries are learnt and programmed to kernel.

   RPD view

```
root@PE1_CRPD> show route table evpn-vxlan.evpn.0 protocol bgp | grep MAC

2:81.2.2.2:1::2600::00:22:22:22:60:00/304 MAC/IP
2:81.2.2.2:1::2601::00:22:22:22:60:10/304 MAC/IP
2:81.2.2.2:1::2600::00:22:22:22:60:00::99.60.0.2/304 MAC/IP
2:81.2.2.2:1::2601::00:22:22:22:60:10::99.60.1.2/304 MAC/IP
2:81.2.2.2:1::2600::00:22:22:22:60:00::1234::633c:2/304 MAC/IP
2:81.2.2.2:1::2600::00:22:22:22:60:00::fe80::5668:a302:5854:1f09/304 MAC/IP
2:81.2.2.2:1::2601::00:22:22:22:60:10::1234::633c:102/304 MAC/IP
2:81.2.2.2:1::2601::00:22:22:22:60:10::fe80::5668:a302:5954:1f0a/304 MAC/IP
```

```
root@PE1_CRPD> show evpn database instance evpn-vxlan origin remote

Instance: evpn-vxlan
```

```
VLAN  DomainId  MAC address        Active source                  Timestamp        IP address
     2600       00:22:22:22:60:00  81.2.2.2                       Apr 29 23:51:56  99.60.0.2

1234::633c:2

fe80::5668:a302:5854:1f09
     2601       00:22:22:22:60:10  81.2.2.2                       Apr 29 23:51:56  99.60.1.2

1234::633c:102

fe80::5668:a302:5954:1f0a
```

Kernel view

Macs are programmed to **bridge fdb** table in Linux

```
root@PE1_CRPD:/# bridge fdb show br __crpd-brd2 dev vxlan2600 state static
```

```
00:22:22:22:60:00 vlan 600 master __crpd-brd193 static
00:00:00:00:00:00 dst 81.2.2.2 self static
00:22:22:22:60:00 dst 81.2.2.2 self static
```

```
root@PE1_CRPD:/# bridge fdb show br __crpd-brd2 dev vxlan2601 state static
```

```
00:22:22:22:60:10 vlan 601 master __crpd-brd193 static
00:00:00:00:00:00 dst 81.2.2.2 self static
00:22:22:22:60:10 dst 81.2.2.2 self static
```

Mac+ip bindings are programmed to **ip neigh** table

```
root@PE1_CRPD:/# ip neigh show dev irb.600 | grep PERMANENT
```

```
99.60.0.2 lladdr 00:22:22:22:60:00 PERMANENT
fe80::5668:a302:5854:1f09 lladdr 00:22:22:22:60:00 PERMANENT
1234::633c:2 lladdr 00:22:22:22:60:00 PERMANENT
```

root@PE1_CRPD:/# **ip neigh show dev irb.601 | grep PERMANENT**

```
99.60.1.2 lladdr 00:22:22:22:60:10 PERMANENT
fe80::5668:a302:5954:1f0a lladdr 00:22:22:22:60:10 PERMANENT
1234::633c:102 lladdr 00:22:22:22:60:10 PERMANENT
```

7

CHAPTER

Best Practices

Security Best Practices

IN THIS SECTION

Following are the best practices required to monitor and secure container environments:

Host OS Hardening

Hardening an operating system inlcudes:

- Ensure that both the host OS and docker software are updated with the latest security patches.
- Download container images that are verified from downloads.
- Run docker as non-root user without root privileges. This is called Rootless mode. In this mode, docker and container run within a user namespace. Running both containers and the Docker Engine services as non-root users improves security in the event of a breach.
- Prevent denial-of-service attacks by configuring specified amount of memory and CPU required to run the containers.
- Avoid using sshd within containers.
- Avoid using default bridge `docker0` from ARP spoofing and MAC flooding attacks.
- Set the container's root filesystem to read-only to prevent from malicious attack.
- Set the process identifier (PID) limit. Each process in the kernel carries a unique PID, and containers leverage Linux PID namespace to provide a separate view of the PID hierarchy for each container. Limiting the number of processes in the container prevents excessive spawning of new processes and potential malicious lateral movement.

Patch Management

Patch management involves identifying system features that can be improved or fixed, releasing the update package, and validating the installation of the updates. Patching with software updates and system reconfiguration is part of vulnerability management.

For information on latest software and details, see downloads and Upgrade cRPD.

Security Ports

Service ports that are priviliged to use are:

- Ensure that only approved BGP port (TCP 179), SSH port (TCP 22), Netconf over SSH (TCP 830) and gRPC ports for telemetry (TCP 50051), protocols, and services with validated business needs are running on each system. For example, HTTP and HTTPS load balancers have to bind (TCP 80) and (TCP 443) respectively.
- TCP/IP port numbers below 1024 are considered privileged ports. Avoid to map any ports below 1024 within a container as they transmit sensitive data. By default, Docker maps container ports to one that's within the 49153–65525 range, but it allows the container to be mapped to a privileged port.