Engineering
Simplicity

Juniper Cloud-Native Router 23.2 Deployment Guide

Published
2025-07-02

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

YEAR 2000 NOTICE

END USER LICENSE AGREEMENT

Table of Contents

1 **Introduction**

Juniper® Cloud-Native Router Overview | 2

Juniper® Cloud-Native Router Components | 4

Cloud-Native Router Deployment Modes | 9

System Requirements | 10

2 **Install**

Before You Install | 21

Customize Cloud-Native Router Helm Chart | 22

Customize Cloud-Native Router Configuration using Node Annotations | 35

Install and Verify Juniper Cloud-Native Router | 42

Install Juniper Cloud-Native Router Using Helm Chart | **42**

Verify Installation | **45**

3 **Monitor**

Monitor Cloud-Native Router via CLI | 52

Telemetry Capabilities of Cloud-Native Router | 59

Logging and Notifications | 66

4 **Manage**

Manage Juniper Cloud-Native Router | 71

5 **Troubleshoot**

Troubleshoot Deployment Issues | 74

Troubleshoot Deployment Issues | **74**

Troubleshoot via the vRouter CLI | 78

Troubleshoot via Introspect | 91

6 Appendix

Working Samples for Cloud-Native Router Helm Chart | 94

Kubernetes Overview | 100

Configure Repository Credentials | 102

Deploy Prepackaged Images | 103

Juniper Technology Previews (Tech Previews) | 104

1

CHAPTER

Introduction

IN THIS CHAPTER

- Juniper® Cloud-Native Router Overview | **2**
- Juniper® Cloud-Native Router Components | **4**
- Cloud-Native Router Deployment Modes | **9**
- System Requirements | **10**

Juniper® Cloud-Native Router Overview

IN THIS SECTION

- Overview | 2
- Use Cases | 2
- Architecture and Key Components | 3
- Features | 4

Overview

While 5G unleashes higher bandwidth, lower latency and higher capacity, it also brings in new infrastructure challenges such as increased number of base stations or cell sites, more backhaul links with larger capacity and more cell site routers and aggregation routers. Service providers are integrating cloud-native infrastructure in distributed RAN (D-RAN) topologies, which are usually small, leased spaces, with limited power, space and cooling. The disaggregation of radio access network (RAN) and the expansion of 5G data centers into cloud hyperscalers has added newer requirements for cloud-native routing.

The Juniper Cloud-Native Router provides the service providers the flexibility to roll out the expansion requirements for 5G rollouts, reducing both the CapEx and OpEx.

Juniper Cloud-Native Router (JCNR) is a containerized router that combines Juniper's proven routing technology with the Junos® containerized routing protocol daemon (cRPD) as the controller and a high-performance Contrail® Data Plane Development Kit (DPDK) vRouter forwarding plane. It is implemented in Kubernetes and interacts seemlessly with a Kubernetes container network (CNI) framework.

Use Cases

The Cloud-Native Router has the following use cases:

- Radio Access Network (RAN)

  The new 5G-only sites are a mix of centralized RAN (C-RAN) and distributed RAN (D-RAN). The C-RAN sites are typically large sites owned by the carrier and continue to deploy physical routers. The D-RAN sites, on the other hand, are tens of thousands of smaller sites, closer to the users.

Optimization of CapEx and OpEx is a huge factor for the large number of D-RAN sites. These sites are also typically leased, with limited space, power and cooling capacities. There is limited connectivity over leased lines for transit back to the mobile core. Juniper Cloud-Native Router is designed to work in the constraints of a D-RAN. It is integrated with the distributed unit (DU) and installable on an existing 1 U server.

- **Telco virtual private cloud (VPC)**

  The 5G data centers are expanding into cloud hyperscalers to support more radio sites. The cloud-native routing available in public cloud environments do not support the routing demands of telco VPCs, such as MPLS, quality of service (QoS), L3 VPN, and more. The Juniper Cloud-Native Router integrates directly into the cloud as a containerized network function (CNF), managed as a cloud-native Kubernetes component, while providing advanced routing capabilities.

Architecture and Key Components

The Juniper Cloud-Native Router consists of the Junos containerized routing protocol Daemon (cRPD) as the control plane (Cloud-Native Router Controller), providing topology discovery, route advertisement and forwarding information base (FIB) programming, as well as dynamic underlays and overlays. It uses the Data Plane Development Kit (DPDK) enabled vRouter as a forwarding plane, providing packet forwarding for DPDK applications in a pod and host path I/O for protocol sessions. The third component is the Cloud-Native Router container network interface (CNI) that interacts with Kubernetes as a secondary CNI to create pod interfaces, assign addresses and generate the cRPD configuration.

The Data Plane Development Kit (DPDK) is an open source set of libraries and drivers. DPDK enables fast packet processing by allowing network interface cards (NICs) to send direct memory access (DMA) packets directly into an application’s address space. The applications poll for packets, to avoid the overhead of interrupts from the NIC. Integrating with DPDK allows a vRouter to process more packets per second than is possible when the vRouter runs as a kernel module.

In this integrated solution, the JCNR Controller uses gRPC, a high performance Remote Procedure Call, based services to exchange messages and to communicate with the vRouter, thus creating the fully functional Cloud-Native Router. This close communication allows you to:

- Learn about fabric and workload interfaces
- Provision DPDK- or kernel-based interfaces for Kubernetes pods as needed
- Configure IPv4 and IPv6 address allocation for Pods
- Install routes into routing tables
- Run routing protocols such as ISIS, BGP, and OSPF

Features

- Easy deployment, removal, and upgrade on general purpose compute devices using Helm
- Higher packet forwarding performance with DPDK-based JCNR-vRouter
- Full routing, switching, and forwarding stacks in software
- Basic L2 functionality, such as MAC learning, MAC aging, MAC limiting, and L2 statistics
- L2 reachability to Radio Units (RU) for management traffic
- L2 or L3 reachability to physical distributed units (DU) such as 5G millimeter wave DUs or 4G DUs
- VLAN tagging
- Bridge domains
- Trunk and access ports
- Support for multiple virtual functions (VF) on Ethernet NICs
- Support for bonded VF interfaces
- Configurable L2 access control lists (ACLs)
- Rate limiting of egress broadcast, unknown unicast, and multicast traffic on fabric interfaces
- IPv4 and IPv6 routing
- Out-of-the-box software-based open radio access network (O-RAN) support
- Quick spin up with containerized deployment
- Highly scalable solution

Juniper® Cloud-Native Router Components

SUMMARY

The Juniper Cloud-Native Router solution consists of several components including the Cloud-Native

IN THIS SECTION

Cloud-Native Router Components | 5

Router controller (cRPD), vRouter and the JCNR-CNI. This topic provides a brief overview of the components of the Juniper Cloud-Native Router.

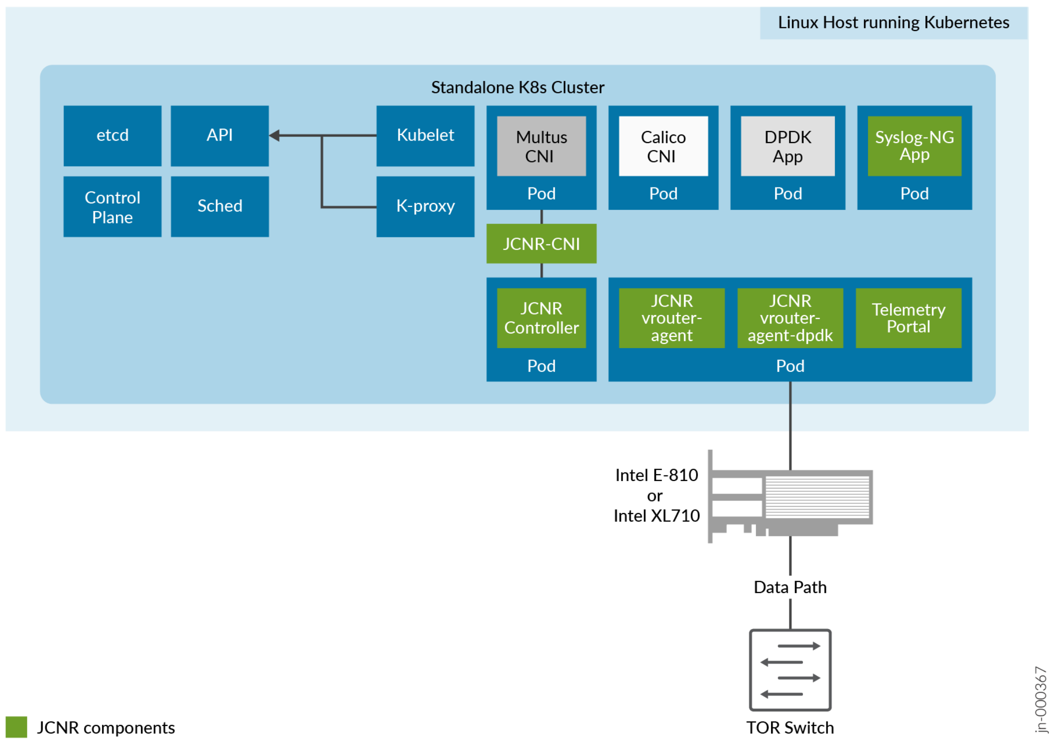
- Cloud-Native Router Controller | 6
- Cloud-Native Router vRouter | 7
- JCNR-CNI | 7
- Syslog-NG | 9

Cloud-Native Router Components

The Juniper Cloud-Native Router has primarily three components—Cloud-Native Router Controller for control plane, the vRouter DPDK forwarding plane and a CNI for Kubernetes integration. All Cloud-Native Router components are deployed as containers.

The Figure 1 on page 5 shows the components of the Juniper Cloud-Native Router inside a Kubernetes cluster

Figure 1: Components of Juniper Cloud-Native Router



Cloud-Native Router Controller

The Cloud-Native Router Controller (cRPD) is the control-plane of the cloud-native router solution and runs as a statefulset. The controller communicates with the other elements of the cloud-native router. Configuration, policies and rules that you set on the controller at deployment time are communicated to other components, primarily the Cloud-Native Router vRouter, for implementation.

For example, firewall filters (ACLs) are supported on cRPD to configure L2 access lists with deny rules. cRPD sends the configuration information to the vRouter through the vRouter agent.

Juniper Cloud-Native Router Controller Functionality:

- Exposes Junos OS compatible CLI configuration and operation commands that are accessible to external automation and orchestration systems using the NETCONF protocol.
- Supports vRouter as the high-speed forwarding plane. This enables applications that are built using the DPDK framework to send and receive packets directly to the application and the vRouter without passing through the kernel.
- Supports configuration of VLAN-tagged sub-interfaces on physical function (PF), virtual function (VF), virtio, access, and trunk interfaces managed by the DPDK-enabled vRouter.
- Supports configuration of bridge domains, VLANs, and virtual-switches.
- Advertises DPDK application reachability to core network using routing protocols primarily with BGP and IS-IS.
- Distributes L3 network reachability information of the pods inside and outside a cluster.
- Maintains configuration for L2 firewall.
- Passes configuration information to the vRouter through the vRouter-agent.
- Stores license key information.
- Works as a BGP Speaker from release 23.2, establishing peer relationships with other BGP speakers to exchange routing information.

Configuration Options

During deployment, you can "customize JCNR using node annotations" on page 35 .

After deployment, we recommend that you use the NETCONF protocol with PyEZ to configure cRPD. Alternatively, you can SSH directly into the cRPD or connect via NETCONF. Finally, you can also configure the cloud-native router by *accessing the JCNR controller (cRPD) CLI* using Kubernetes commands.

Cloud-Native Router vRouter

The Cloud-Native Router vRouter is a high-performance datapath component. It is an alternative to the Linux bridge or the Open vSwitch (OVS) module in the Linux kernel. It runs as a user-space process and is integrated with the Data Plane Development Kit (DPDK) library. The vRouter pod consists of three containers—vrouter-agent, vrouter-agent-dpdk and vrouter-telemetry-exporter.

Cloud-Native Router vRouter Functionality:

- Performs routing with Layer 3 virtual private networks
- Performs L2 forwarding
- Allows the use of DPDK-based forwarding
- Enforces L2 access control lists (ACLs)

Benefits of vRouter:

- Integration of the DPDK into the JCNR-vRouter
- Forwarding plane provides faster forwarding capabilities than kernel-based forwarding
- Forwarding plane is more scalable than kernel-based forwarding
- Support for the following NICs:
  - Intel E810 (Columbiaville) family
  - Intel XL710 (Fortville) family

JCNR-CNI

JCNR-CNI is a new container network interface (CNI) developed by Juniper. JCNR-CNI is a Kubernetes CNI plugin installed on each node to provision network interfaces for application pods. During pod creation, Kubernetes delegates pod interface creation and configuration to JCNR-CNI. JCNR-CNI interacts with Cloud-Native Router control-plane (cRPD) and the vRouter to setup DPDK interfaces. When a pod is removed, JCNR-CNI is invoked to de-provision the pod interface, configuration, and associated state in Kubernetes and cloud-native router components. JCNR-CNI works as a secondary CNI, along with the Multus CNI to add and configure pod interfaces.

JCNR-CNI Functionality:

- Manages the networking tasks in Kubernetes pods such as:
  - assigning IP addresses

- allocating MAC addresses
- setting up untagged, access, and other interfaces between the pod and vRouter in a Kubernetes cluster
- creating VLAN sub-interfaces

- Acts on pod events such as add and delete
- Generates cRPD configuration

The JCNR-CNI manages the secondary interfaces that the pods use. It creates the required interfaces based on the configuration in YAML-formatted network attachment definition (NAD) files. The JCNR-CNI configures some interfaces before passing them to their final location or connection point and provides an API for further interface configuration options such as:

- Instantiating different kinds of pod interfaces.
- Creating virtio-based high performance interfaces for pods that leverage the DPDK data plane.
- Creating veth pair interfaces that allow pods to communicate using the Linux Kernei l networking stack.
- Creating pod interfaces in access or trunk mode.
- Attaching pod interfaces to bridge domains.
- Supporting IPAM plug-in for Dynamic IP address allocation.
- Allocating unique socket interfaces for virtio interfaces.
- Attaching pod interfaces to a bridge domain.
- Managing the networking tasks in pods such as assigning IP addresses and setting up of interfaces between the pod and vRouter in ai Kubernetes cluster.
- Connecting pod interface to a network including pod-to-pod and pod-to-network.
- Integrating with the vRouter for offloading packet processing.

Benefits of JCNR-CNI:

- Improved pod interface management
- Customizable administrative and monitoring capabilities
- Increased performance through tight integration with cRPD and vRouter components

The Role of JCNR-CNI in Pod Creation:

When you create a pod for use in the cloud-native router, the Kubernetes component known as **kubelet** calls the Multus CNI to set up pod networking and interfaces. Multus reads the annotations section of the **pod.yaml** file to find the NADs. If a NAD points to JCNR-CNI as the CNI plug in, Multus calls the JCNR-CNI to set up the pod interface. JCNR-CNI creates the interface as specified in the NAD. JCNR-CNI then generates and pushes a configuration into cRPD.

Syslog-NG

Juniper Cloud-Native Router uses a syslog-ng pod to gather event logs from cRPD and vRouter and transform the logs into JSON-based notifications. The notifications are logged to a file. Syslog-ng runs as a daemonset.

Cloud-Native Router Deployment Modes

SUMMARY

Read this topic to know about the various modes of deploying the cloud-native router.

IN THIS SECTION

- Deployment Modes | 9

Deployment Modes

Starting with Juniper Cloud-Native Router Release 23.2, you can deploy and operate Juniper Cloud-Native Router in L2, L3 and L2-L3 modes*, auto-derived based on the interface configuration in the `values.yaml` file prior to deployment. Please review the "Customize Cloud-Native Router Helm Chart" on page 22 topic for more information.

NOTE: In the `values.yaml` file:

- When all the interfaces have an `interface_mode` key configured, then the mode of deployment would be L2.

- When one or more interfaces have an `interface_mode` key configured and some of the interfaces do not have the `interface_mode` key configured, then the mode of deployment would be L2-L3*.

- When none of the interfaces have the `interface_mode` key configured, then the mode of deployment would be L3.

In L2 mode, the cloud-native router behaves like a switch and therefore does not performs any routing functions and it doesn not run any routing protocols. The pod network uses VLANs to direct traffic to various destinations.

In L3 mode, the cloud-native router behaves like a router and therefore performs routing functions and runs routing protocols such as ISIS, BGP, OSPF, and segment routing-MPLS. In L3 mode, the pod network is divided into an IPv4 or IPv6 underlay network and an IPv4 or IPv6 overlay network. The underlay network is used for control plane traffic.

The L2-L3 mode* provides the functionality of both the switch and the router at the same time. It enables Cloud-Native Router to act as both a switch and a router simultaneously by performing switching in a set of interfaces and routing in the other set of interfaces. Cell site routers in a 5G deployment need to handle both L2 and L3 traffic. DHCP packets from radio outdoor unit (RU) is an example of L2 traffic and data packets moving from outdoor unit (ODU) to central unit (CU) is an example of L3 traffic.

NOTE: *The L2-L3 deployment mode is a *Juniper Technology Previews (Tech Previews)* feature in the Juniper Cloud-Native Router Release 23.2.

System Requirements

IN THIS SECTION

- Host System Requirements | 11
- Resource Requirements | 13
- Miscellaneous Requirements | 14
- Port Requirements | 17
- Cloud-Native Router Licensing | 18

Read this section to understand the system, resource, port, and licensing requirements for installing Juniper Cloud-Native Router.

Host System Requirements

This section lists the host system requirements for installing the cloud-native router.

Table 1: Cloud-Native Router Host System Requirements

	Release 23.2	
Component	Value/Version	Notes
CPU	Intel x86	The tested CPU is Intel Xeon Gold 6212U 24-core @2.4 GHz
Host OS	RedHat Enterprise Linux	Version 8.4, 8.5, 8.6
	Rocky Linux	8.6
Kernel Version	RedHat Enterprise Linux (RHEL): 4.18.X Rocky Linux: 4.18.X	The tested kernel version for RHEL is 4.18.0-305.rt7.72.el8.x86_64 The tested kernel version for Rocky Linux is 4.18.0-372.19.1.rt7.176.el8_6.x86_64 and 4.18.0-372.32.1.rt7.189.el8_6.x86_64

Table 1: Cloud-Native Router Host System Requirements *(Continued)*

Component	Release 23.2	
	Value/Version	Notes
NIC	• Intel E810 with Firmware 4.00 0x80014411 1.3236.0 • Intel E810-CQDA2 with Firmware 4.000x800144111.3236.0 • Intel XL710 with Firmware 9.00 0x8000cead 1.3179.0 • Elastic Network Adapter (ENA)	
IAVF driver	Version 4.5.3.1	
ICE_COMMS	Version 1.3.35.0	
ICE	Version 1.9.11.9	ICE driver is used only with the Intel E810 NIC
i40e	Version 2.18.9	i40e driver is used only with the Intel XL710 NIC
Kubernetes (K8s)	Version 1.22.x, 1.23.x, 1.25x	The tested K8s version is 1.22.4, although 1.22.2 will also work.
Calico	Version 3.22.x	
Multus	Version 3.8	

Table 1: Cloud-Native Router Host System Requirements *(Continued)*

Component	Release 23.2	
	Value/Version	Notes
Helm	3.9.x	
Container-RT	Docker CE 20.10.11, crio 1.25x	
Amazon EKS		The K8s version is 1.23.17. Kernel version is 5.4.235-144.344.amzn2.x86_64 OS version is Amazon Linux 2
OpenShift	4.12.0	The K8s version is 1.25.4.

Resource Requirements

This section lists the resource requirements for installing the cloud-native router.

Table 2: Cloud-Native Router Resource Requirements

Resource	Release 23.2	
	Value	Usage Notes
Data plane forwarding cores	2 physical cores (2p)	
Service/Control Cores	0	
UIO Driver	VFIO-PCI	

Table 2: Cloud-Native Router Resource Requirements *(Continued)*

Resource	Release 23.2	
	Value	Usage Notes
Hugepages (1G)	6 Gi	Add `GRUB_CMDLINE_LINUX_DEFAULT` values in **/etc/default/grub** and reboot the host. For example: `GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
Cloud-Native Router Controller cores	.5	
Cloud-Native Router vRouter Agent cores	.5	

Miscellaneous Requirements

This section lists additional requirements for installing the cloud-native router.

Table 3: Miscellaneous Requirements

Cloud-Native Router Release 23.2 Miscellaneous Requirements
Enable VLAN driver at system boot using the command: `modprobe 8021q` Verify by executing the command `lsmod | grep 8021q`
Enable VFIO-PCI driver at system boot

Table 3: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release 23.2 Miscellaneous Requirements

Set IOMMU and IOMMU-PT in `/etc/default/grub` file. For example:
 	
`GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
 	
Update grub and reboot the host. For example:
 	
`grub2-mkconfig -o /boot/grub2/grub.cfg`

Disable Spoofcheck on VFs allocated to JCNR. For example: `ip link set <interfacename> vf 1 spoofcheck off`. **NOTE**: Applicable only on L2 deployments.
Set trust on VFs allocated to JCNR. For example: `ip link set <interfacename> vf 1 trust on` **NOTE**: Applicable only on L2 deployments.
Additional kernel modules need to be loaded on the host before deploying Cloud-Native Router in L3 mode. These modules are usually available in `linux-modules-extra` or `kernel-modules-extra` packages. Run the following commands to add the kernel modules. • modprobe tun • modprobe fou • modprobe fou6 • modprobe ipip • modprobe ip_tunnel • modprobe ip6_tunnel • modprobe mpls_gso • modprobe mpls_router • modprobe mpls_iptunnel • modprobe vrf • modprobe vxlan

Table 3: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release 23.2 Miscellaneous Requirements
Run the `ip fou add port 6635 ipproto 137` command on the Linux host to enable kernel based forwarding.

NetworkManager is a tool in some operating systems to make the management of network interfaces easier. NetworkManager may make the operation and configuration of the default interfaces easier. However, it can interfere with the Kubernetes management and create problems.
 	
To avoid the NetworkManager from interfering with the interface configurations, perform the following steps:
 	
1. Create the file, `/etc/NetworkManager/conf.d/crpd.conf`.
2. Add the following content in the file.
 	
`[keyfile]`
` unmanaged-devices+=interface-name:enp*;interface-name:ens*`
 	
NOTE: enp* indicates all interfaces starting with enp. For specific interface names, provided a comma-separated list.
 	
3. Restart the NetworkManager service by running the command, `sudo systemctl restart NetworkManager`.
4. Edit the `sysctl` file on the host and paste the following content in it:
 	
`net.ipv6.conf.default.addr_gen_mode=0`
`net.ipv6.conf.all.addr_gen_mode=0`
`net.ipv6.conf.default.autoconf=0`
`net.ipv6.conf.all.autoconf=0`
 	
5. Run the command `sysctl -p /etc/sysctl.conf` to load the new `sysctl.conf` values on the host.
6. Create the bond interface manually. For example:
 	
`ifconfig ens2f0 down`
`ifconfig ens2f1 down`
`ip link add bond0 type bond mode 802.3ad`
`ip link set ens2f0 master bond0`
`ip link set ens2f1 master bond0`
`ifconfig ens2f0 up ; ifconfig ens2f1 up; ifconfig bond0 up`

Table 3: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release 23.2 Miscellaneous Requirements

Verify the core_pattern value is set on the host before deploying JCNR:
`sysctl kernel.core_pattern`
`kernel.core_pattern = |/usr/lib/systemd/systemd-coredump %P %u %g %s %t %c %h %e`
You can update the core_pattern in `/etc/sysctl.conf`. For example:
`kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`

Port Requirements

Juniper Cloud-Native Router listens on certain TCP and UDP ports. This section lists the port requirements for the cloud-native router.

Table 4: Cloud-Native Router Listening Ports

Protocol	Port	Description
TCP	8085	vRouter introspect–Used to gain internal statistical information about vRouter
TCP	8070	Telemetry information-Used to see telemetry data from cloud-native router
TCP	9091	vRouter health check–cloud-native router checks to ensure **contrail-vrouter-dpdk** process is running, etc.
TCP	50052	gRPC port–Cloud-Native Router listens on both IPv4 and IPv6
TCP	24	cRPD SSH

Table 4: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	830	cRPD NETCONF
TCP	666	**rpd**
TCP	1883	Mosquito mqtt–Publish/subscribe messaging utility
TCP	9500	**agentd** on cRPD
TCP	21883	**na-mqttd**
TCP	50051	**jsd** on cRPD
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Cloud-Native Router Licensing

Starting with Juniper Cloud-Native Router (JCNR) Release 22.2, we have enabled our Juniper Agile Licensing (JAL) model. JAL ensures that features are used in compliance with Juniper's end-user license agreement. You can purchase licenses for the Juniper Cloud-Native Router software through your Juniper Account Team. You can apply the licenses by using the CLI of the cloud-native router controller. For details about managing multiple license files for multiple cloud-native router deployments, see Juniper Agile Licensing Overview.

NOTE: Starting with Cloud-Native Router Release 23.2, the Cloud-Native Router license format has changed. Request a new license key from the JAL portal before deploying or upgrading to 23.2 or newer releases.

To verify your Cloud-Native Router license:

1. Run the command `kubectl get pods -A | grep -i crpd` on the host server.

   ```
   jcnr      kube-crpd-worker-sts-0      1/1     Running   0            24h
   ```

2. Identify the cRPD pod and issue the command `kubectl exec -it -n jcnr kube-crpd-worker-sts-0 -- cli show system license`.

   ```
   Defaulted container "kube-crpd-worker" out of: kube-crpd-worker, jcnr-crpd-config (init),
   install-cni (init)
   License usage:
                                    Licensed     Licensed    Licensed
                                     Feature      Feature     Feature
     Feature name                       used    installed      needed    Expiry
     containerized-rpd-standard            1            1           0    2026-02-04 16:00:00 PST

   Licenses installed:
     License identifier: JUNOS892191212
     License version: 4
     Order Type: commercial
     Features:
       containerized-rpd-standard - Containerized routing protocol daemon with standard features
         date-based, 2023-02-05 16:00:00 PST - 2026-02-04 16:00:00 PST
   ```

2

CHAPTER

Install

IN THIS CHAPTER

- Before You Install | 21
- Customize Cloud-Native Router Helm Chart | 22
- Customize Cloud-Native Router Configuration using Node Annotations | 35
- Install and Verify Juniper Cloud-Native Router | 42

Before You Install

SUMMARY

This topic provides a list of pre-requisities for installing the Juniper Cloud-Native Router including the required host configuration and licenses.

The Juniper Cloud-Native Router (JCNR) is a container-based software solution, orchestrated by Kubernetes(K8s). It combines the containerized routing protocol process (cRPD) and a Data Plane Development Kit (DPDK)-enabled virtual router (vRouter). Using the JCNR, you can enable full Junos routing control plane with enhanced forwarding capabilities.

This section provides a list of pre-requisites for installing JCNR.

1. Review the JCNR "System Requirements" on page 10 to setup and configure the host machines.
2. Cloud-Native Router supports an all-in-one or multinode Kubernetes cluster, with master and worker nodes running on virtual machines (VMs) or bare metal servers (BMS). Kubernetes installation or management instructions are not provided in this documentation. Please see https://kubernetes.io for Kubernetes documentation.

   NOTE: It is not recommended to deploy Juniper Cloud-Native Router if Kubernetes cpumanager is enabled in your Kubernetes cluster.

3. Ensure the host is enabled with SR-IOV and VT-d in the system's BIOS.
4. Hugepage and iommu setting is mandatory for the vRouter to come up. You can set them at grub level with the below options:

   a. Add below params by adjusting the no of hugepages (64 here is just an example) in `/etc/default/grub` and reboot the host: "

   ```
   GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G
   hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt
   ```

   Update grub and reboot the host. For example:

   ```
   grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

b. To verify hugepage is set , execute these commands on the host:

```
cat /proc/cmdline
grep -i hugepages /proc/meminfo
```

5. Ensure VLAN driver is enabled on the host machine at startup. You can verify it by issuing the `lsmod | grep 8021q` command.
6. Enable virtual functions (VFs) on host NICs at startup. (Only for L2 deployments)
7. Enable Trust and Disable Spoofcheck on VF's assigned to vRouter using the commands (Only for L2 deployments):

```
ip link set <interfacename> vf 1 spoofchk off
ip link set <interfacename> vf 1 trust on
```

8. Ensure you have the latest version of `helm3` installed.
9. Download the Cloud-Native Router tarball, **Juniper_Cloud_Native_Router_*release_nunber*.tgz** from the Juniper Support Site, to a directory on your host machine. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
10. Ensure you have a valid cRPD license available. The Cloud-Native Router license format has changed, you must request a new license key before deploying or upgrading to 23.2 release. See "Cloud-Native Router Licensing" on page 18 for more information.
11. Review the *Cloud-Native Router Deployment Modes* topic to understand the various deployment modes supported by JCNR.

Customize Cloud-Native Router Helm Chart

Read this topic to learn about the deployment configuration available for the Juniper Cloud-Native Router.
You can deploy and operate Juniper Cloud-Native Router in the L2, L3, or L2-L3* mode. You configure the deployment mode by editing the appropriate attributes in the `values.yaml` file prior to deployment.

NOTE:

- The L2-L3 deployment mode is a "Juniper Technology Previews (Tech Previews)" on page 104 feature in the Juniper Cloud-Native Router Release 23.2.
- In the `fabricInterface` key of the `values.yaml` file:
  - When all the interfaces have an `interface_mode` key configured, then the mode of deployment would be L2.
  - When one or more interfaces have an `interface_mode` key configured along with the rest of the interfaces not having the `interface_mode` key, then the mode of deployment would be L2-L3.
  - When none of the interfaces have the `interface_mode` key configured, then the mode of deployment would be L3.

Customize the helm charts using the `Juniper_Cloud_Native_Router_`*`version-number`*`/helmchart/values.yaml` file. The configuration keys of the heml chart are shown in the table below.

Table 5: Helm Chart Attributes and Descriptions

Key	Additional Key Configuration	Description
registry		Defines the docker registry where the vRouter, cRPD and jcnr-cni container images are hosted. The default value is `enterprise-hub.juniper.net`.
repository		(Optional) Defines the repository path for the vRouter, cRPD and jcnr-cni container images. This is a global key and takes precedence over "repository" paths under "common" section.
imagePullSecret		(Optional) Defines the registry authentication credentials. You can configure credentials to either the Juniper repository or your private registry.
	registryCredentials	Base64 representation of your Docker registry credentials. View the "Configure Repository Credentials" on page 102 topic for more information.
	secretName	Name of the Secret object that will be created.
common		Defines repsitory paths and tags for the vRouter, cRPD and jcnr-cni container images.

Table 5: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
	repository	Defines the repository path. The default value is `atom-docker/cn2/bazel-build/dev/`. The global repository key takes precedence if defined.
	tag	Defines the image tag. The default value is configured to the appropriate tag number for the Cloud-Native Router release version.
replicas		(Optional) Indicates the number of replicas for cRPD. If the value is not specified, then the default value 1 is considered. The value for this key must be specified for multi-node clusters.
storageClass		(Optional) Indicates the name of the storage class for cRPD. This key must be specified for cloud deployments such as AWS where gp2 can be used.
noLocalSwitching		(Optional) Prevents interfaces in a bridge domain from transmitting and receiving ethernet frame copies. Enter one or more comma separated VLAN IDs to ensure that the interfaces belonging to the VLAN IDs do not transmit frames to one another. This key is specific for L2 and L2-L3 deployments. Enabling this key provides the functionality on all access interfaces. For enabling the functionality on trunk interfaces, configure the no-local-switching key in the fabricInterface key.

Table 5: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description

fabricInterface		Provide a list of interfaces to be bound to the DPDK. You can also provide subnets instead of interface names. If both the interface name and the subnet are specified, then the interface name takes precedence over subnet/gateway combination. The subnet/ gateway combination is useful when the interface names vary in a multi-node cluster.
		
		NOTE:
		• When all the interfaces have an `interface_mode` key configured, then the mode of deployment would be L2.
		• When one or more interfaces have an `interface_mode` key configured along with the rest of the interfaces not having the `interface_mode` key, then the mode of deployment would be L2-L3.
		• When none of the interfaces have the `interface_mode` key configured, then the mode of deployment would be L3.
		
		For example:
		
		`# L2 only`
		`- eth1:`
		`    ddp: "auto"`
		`    interface_mode: trunk`
		`    vlan-id-list: [100, 200, 300, 700-705]`
		`    storm-control-profile: rate_limit_pf1`
		`    native-vlan-id: 100`
		`    no-local-switching: true`
		`- bond0:`
		`    ddp: "auto" # auto/on/off`
		`    interface_mode: trunk`
		`    vlan-id-list: [100, 200, 300, 700-705]`
		`    storm-control-profile: rate_limit_pf1`
		`    #native-vlan-id: 100`
		`    #no-local-switching: true`
		
		`# L3  only`
		`- eth1:`
		`    ddp: "off"`

Table 5: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
		```
		  - eth2:
		      ddp: "off"

		 # L2L3
		  - eth1:
		      ddp: "auto"
		  - eth2:
		      ddp: "auto"
		      interface_mode: trunk
		      vlan-id-list: [100, 200, 300, 700-705]
		      storm-control-profile: rate_limit_pf1
		      native-vlan-id: 100
		      no-local-switching: true
		```

	subnet	An alternative mode of input for interface names. For example:
		```
		- subnet: 10.40.1.0/24
		  gateway: 10.40.1.1
		  ddp: "off"
		```
		The `subnet` option is applicable only for L3 interfaces. With the `subnet` mode of input, interfaces are auto-detected in each subnet. Specify either subnet/gateway or the interface name. Do not configure both. The subnet/gateway form of input is particularly helpful in environments where the interface names vary for a multi-node K8s cluster.

	ddp	(Optional) Indicates the interface-level Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at NIC for traffic like GTPU, SCTP, etc. For a bond interface, all slave interface NICs must support DDP for the DDP configuration to be enabled. Setting options include auto, on, or off. The default setting is off. **NOTE**: The subnet/interface level `ddp` takes precedence over the global `ddp` configuration.

Table 5: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
	interface_mode	Set to `trunk` for L2 interfaces and **do not** configure for L3 interfaces. For example, `interface_mode: trunk`
	vlan-id-list	Provide a list of VLAN IDs associated with the interface.
	storm-control-profile	Use `storm-control-profile` to associate appropriate storm control profile for the interface. Profiles are defined under `jcnr-vrouter.stormControlProfiles`.

	native-vlan-id	Configure `native-vlan-id` with any of the VLAN IDs in the vlan-id-list to associate it with untagged data packets received on the physical interface of a fabric trunk mode interface. For example:
		
		`fabricInterface:`
		`  - bond0:`
		`      interface_mode: trunk`
		`      vlan-id-list: [100, 200, 300]`
		`      storm-control-profile: rate_limit_pf1`
		`      native-vlan-id: 100`

	no-local-switching	Prevents interfaces from communicating directly with each other if the no-local-switching statement is configured. Allowed values are true or false.
fabricWorkloadInterface		(Optional) Defines the interfaces to which different workloads are connected. They can be software-based or hardware-based interfaces.
log_level		Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave the log_level set to INFO unless instructed to change it by Juniper support.

Table 5: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
log_path		The defined directory stores various Cloud-Native Router related descriptive logs such as contrail-vrouter-agent.log, contrail-vrouter-dpdk.log, etc.
syslog_notifications		Indicates the absolute path to the file that stores syslog-ng generated notifications in JSON format.

nodeAffinity		(Optional) Defines labels on nodes to determine where to place the cRPD, vRouter and syslog-ng pods. For example:
		`nodeAffinity:`
		`  - key: node-role.kubernetes.io/worker`
		`    operator: Exists`
		NOTE: This key is a global setting.
		On an OCP setup node affinity must be configured to bring up Cloud-Native Router on worker only.

	key	Key-value pair that represents a node label that must be matched to apply the node affinity.
	operator	Defines the relationship between the node label and the set of values in the matchExpression parameters in the pod specification. This value can be In, NotIn, Exists, DoesNotExist, Lt, or Gt.
cni_bin_dir		(Optional) The default path is /opt/cni/bin. You can override the default cni path with a path of your choice e.g. /var/opt/cni/bin. In some deployments like Red Hat OpenShift the default CNI path may need to be changed. Leaving the path variable (cni_bin_dir) empty, isn't a viable option in OCP.
grpcTelemetryPort		(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50051.
grpcVrouterPort		(Optional) Enter a value for this parameter to override vRouter gRPC server default port of 50052.

Table 5: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
restoreInterfaces		Set the value of this key to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts.
bondInterfaceConfigs		(Optional) Enable bond interface configurations only for L2 or L2-L3 deployments.
	name	(Optional) Name of the bond interface.
	mode	(Optional) Default value is 1 (Active_Backup)
	slaveInterfaces	(Optional)
mtu		Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs).
cpu_core_mask		Indicates the vRouter forward core mask. If qos is enabled, you will need to allocate 4 CPU cores (primary and siblings).
stormControlProfiles		Configure the rate limit profiles for BUM traffic on fabric interfaces in bytes per second.
	rate_limit_pf1	
	bandwidth	
	level	
ddp		(Optional) Indicates the global Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at NIC for traffic like GTPU, SCTP, etc. For a bond interface, all slave interface NICs must support DDP for the DDP configuration to be enabled. Setting options include auto, on, or off. The default setting is off. **NOTE**: The subnet/interface level `ddp` takes precedence over the global `ddp` configuration.

Table 5: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
qosEnable		Set to true or false to enable or disable QoS. **NOTE**: QoS is not supported on Intel X710 NIC.
corePattern		Indicates the core pattern to denote how the core file is generated. If this configuration is left blank, then Cloud-Native Router pods will not overwrite the default pattern. **NOTE**: Set the corePattern value on host before deploying JCNR. You may change the value in `/etc/sysctl.conf`. For example, `kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`
coreFilePath		Indicates the path for the core file. If the value is left blank, then vRouter considers /var/crashes as the default value.
vrouter_dpdk_uio_driver		The uio driver can either be `vfio-pci` or `uio_pci_generic`.

The default helm chart is shown below:

```
########################################################################
#                 Common Configuration (global vars)               #
########################################################################
global:
  registry: enterprise-hub.juniper.net/
  # uncomment below if all images are available in the same path; it will
  # take precedence over "repository" paths under "common" section below
  repository: jcnr-container-prod/

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
  #   registryCredentials: <base64-encoded-credential>
  #   secretName: regcred
  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/
      tag: R23.2-156
    crpd:
      repository: junos-docker-local/warthog/
      tag: 23.2R1.14
    jcnrcni:
      repository: junos-docker-local/warthog/
      tag: 23.2-20230628-8cf4350

  # Number of replicas for cRPD; this option must be used for multinode clusters
  # JCNR will take 1 as default if replicas is not specified
  #replicas: "3"

  # storageClass: Name of the storage class for cRPD. This option is must for
  # cloud deployments such as AWS where gp2 can be used
  #storageClass: gp2

  #noLocalSwitching: [700]

  # fabricInterface: provide a list of interfaces to be bound to dpdk
  # You can also provide subnets instead of interface names. Interfaces name take precedence over
  # Subnet/Gateway combination if both specified (although there is no reason to specify both)
  # Subnet/Gateway combination comes handy when the interface names vary in a multi-node cluster
  fabricInterface:
  #########################
  # L2 only
  #- eth1:
  #    ddp: "auto"                # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  #- eth2:
  #    ddp: "auto"                # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [700]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  #- bond0:
  #    ddp: "auto" # auto/on/off  # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    #native-vlan-id: 100
  #    #no-local-switching: true

  #########################
  # L3  only
  #- eth1:
  #    ddp: "off"                 # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "off"                 # ddp parameter is optional; options include auto or on or
off; default: off
  ########################

  # L2L3
  #- eth1:
  #    ddp: "auto"                # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "auto"                # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  ##################################

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
  #- subnet: 10.40.1.0/24
  #  gateway: 10.40.1.1
  #  ddp: "off"                     # ddp parameter is optional; options include auto or on or
```

```
off; default: off
  #- subnet: 192.168.1.0/24
  #  gateway: 192.168.1.1
  #  ddp: "off"                     # ddp parameter is optional; options include auto or on or
off; default: off

  ##################################
  # fabricWorkloadInterface is applicable only for Pure L2 deployments
  #
  #fabricWorkloadInterface:
  #- enp59s0f1v0:
  #    interface_mode: access
  #    vlan-id-list: [700]
  #- enp59s0f1v1:
  #    interface_mode: trunk
  #    vlan-id-list: [800, 900]
 #########################

  # defines the log severity. Possible options: DEBUG, INFO, WARN, ERR
  log_level: "INFO"

  # "log_path": this directory will contain various jcnr related descriptive logs
  # such as contrail-vrouter-agent.log, contrail-vrouter-dpdk.log etc.
  log_path: "/var/log/jcnr/"
  # "syslog_notifications": absolute path to the file that will contain syslog-ng
  # generated notifications in json format
  syslog_notifications: "/var/log/jcnr/jcnr_notifications.json"

  # nodeAffinity: Can be used to inject nodeAffinity for vRouter, cRPD and syslog-ng pods
  # You may label the nodes where we wish to deploy JCNR and inject affinity accodingly
  #nodeAffinity:
  #- key: node-role.kubernetes.io/worker
  #  operator: Exists
  #- key: node-role.kubernetes.io/master
  #  operator: DoesNotExist
  #- key: kubernetes.io/hostname
  #  operator: In
  #  values:
  #  - example-host-1

  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
```

```
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  #cni_bin_dir:  /var/lib/cni/bin

  # grpcTelemetryPort: use this parameter to override cRPD telemetry gRPC server default port of 
50051
  #grpcTelemetryPort: 50055

  # grpcVrouterPort: use this parameter to override vRouter gRPC server default port of 50052
  #grpcVrouterPort: 50060

jcnr-vrouter:
  # restoreInterfaces: setting this to true will restore the interfaces
  # back to their original state in case vrouter pod crashes or restarts
  restoreInterfaces: false

 # Enable bond interface configurations L2 only or L2 L3 deployment

  #bondInterfaceConfigs:
  #  - name: "bond0"
  #    mode: 1             # ACTIVE_BACKUP MODE
  #    slaveInterfaces:
  #    - "enp59s0f0v0"
  #    - "enp59s0f0v1"

  # MTU for all physical interfaces( all VF’s and  PF’s)
  mtu: "9000"

  # vrouter fwd core mask
  # if qos is enabled, you will need to allocate 4 CPU cores (primary and siblings)
  cpu_core_mask: "2,3,22,23"

  # rate limit profiles for bum traffic on fabric interfaces in bytes per second
  stormControlProfiles:
    rate_limit_pf1:
      bandwidth:
        level: 0
    #rate_limit_pf2:
    #  bandwidth:
    #    level: 0

  # Set ddp to enable Dynamic Device Personalization (DDP)
  # Provides datapath optimization at NIC for traffic like GTPU, SCTP etc.
  # Options include auto or on or off; default: off
```

```
ddp: "auto"

# Set true/false to Enable or Disable QOS, note: QOS is not supported on X710 NIC.
qosEnable: false

# core pattern to denote how the core file will be generated
# if left empty, JCNR pods will not overwrite the default pattern
corePattern: ""

# path for the core file; vrouter considers /var/crashes as default value if not specified
coreFilePath: /var/crash

# uio driver will be vfio-pci or uio_pci_generic
vrouter_dpdk_uio_driver: "vfio-pci"
```

NOTE: If you are installing Cloud-Native Router on Amazon EKS, then update the `dpdkCommandAdditionalArgs` key in the **helmchart/charts/jcnr-vrouter/values.yaml** file and set `tx` and `rx` descriptors to 256. For example:

```
dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 256 --dpdk_rxd_sz 256"
```

Additional working samples for various deployment modes have been provided in the "Working Samples for Cloud-Native Router Helm Chart" on page 94 topic.

Customize Cloud-Native Router Configuration using Node Annotations

SUMMARY

Read this topic to understand how to customize Cloud-Native Router configuration using node annotations and custom configuration template.

IN THIS SECTION

- Node Annotations | 36
- Configuration Example | 36
- Troubleshooting | 42

Node Annotations

Starting with Juniper Cloud-Native Router (JCNR) Release 23.2, Cloud-Native Router supports customizing the Cloud-Native Router configurations using node annotations when deployed in L3 mode. Node annotations are key-value pairs. The key-value pairs are used to render the cRPD configuration via a go template. The configured template must be available in the `Juniper_Cloud_Native_Router_`*`release_number`*`/helmchart/charts/jcnr-cni/files/` directory for the configuration to be applied to the cRPD pods.

NOTE: You must apply the node annotations before installing Cloud-Native Router to create cRPD pods with custom configuration. The cRPD pod must be deleted and respawned should you wish to apply the annotations any time after Cloud-Native Router installation. cRPD customization via node annotations is optional.

Configuration Example

Sample node annotation and template files are available under `Juniper_Cloud_Native_Router_`*`<release-number>`*`/helmchart/cRPD_examples` directory.

You define the key-value pair for each worker node of your cluster. An example of the `node-annotations.yaml` file is provided below:

```
apiVersion: v1
kind: Node
metadata:
  name: 5d8s1-node1                                           ---> Node name
  annotations:
    jcnr.juniper.net/params: '{
        "isoLoopbackAddr": "49.0004.1000.0000.0001.00",       ---> Key value pairs
        "IPv4LoopbackAddr": "110.1.1.2",
        "srIPv4NodeIndex": "2000",
        "srIPv6NodeIndex": "3000",
        "BGPIPv4Neighbor": "110.1.1.254",
        "BGPLocalAsn": "64512"
    }'
---
apiVersion: v1
kind: Node
metadata:
  name: 5d8s1-node2
  annotations:
    jcnr.juniper.net/params: '{
        "isoLoopbackAddr": "49.0004.1000.0000.0000.00",
        "IPv4LoopbackAddr": "110.1.1.3",
        "srIPv4NodeIndex": "2001",
        "srIPv6NodeIndex": "3002",
        "BGPIPv4Neighbor": "110.1.2.254",
        "BGPLocalAsn": "64512"
    }'
```

The key-value pairs you define in the annotations is used to render the cRPD configuration via a go template. An example of the `jcnr-cni-custom-config.tmpl` template file is provided below:

```

    apply-groups [custom];
    groups {
        custom {
            interfaces {
                lo0 {
                    unit 0 {
                        {{if .Node.isoLoopbackAddr}}               ---> key is replaced by
value, if available for the node
                        family iso {
                            address {{.Node.isoLoopbackAddr}};
                        }
                        {{end}}
                        family inet {
                            address {{.Node.IPv4LoopbackAddr}};
                        }
                    }
                }
            }
            routing-options {
               router-id {{.Node.IPv4LoopbackAddr}}
               route-distinguisher-id {{.Node.IPv4LoopbackAddr}}
            }
            protocols {
                isis {
                   interface all;
                   {{if and .Env.SRGB_START_LABEL .Env.SRGB_INDEX_RANGE}}    ---> conditional
statement based on environment variables
                   source-packet-routing {
                       srgb start-label {{.Env.SRGB_START_LABEL}} index-range
{{.Env.SRGB_INDEX_RANGE}};
```

```
           node-segment {
               {{if .Node.srIPv4NodeIndex}}
               ipv4-index {{.Node.srIPv4NodeIndex}};
               {{end}}
               {{if .Node.srIPv6NodeIndex}}
               ipv6-index {{.Node.srIPv6NodeIndex}};
               {{end}}
           }
       }
       {{end}}
       level 1 disable;
    }
    ldp {
        interface all;
    }
    mpls {
        interface all;
    }
    bgp {
        interface all;
    }
}
policy-options {
    # policy to signal dynamic UDP tunnel attributes to BGP routes
    policy-statement udp-export {
        then community add udp;
    }
    community udp members encapsulation:0L:13;
}
protocols {
    bgp {
        group jcnrbgp1 {
            type internal;
            local-address {{.Node.IPv4LoopbackAddr}};
            local-as {{.Node.BGPLocalAsn}};
            neighbor {{.Node.BGPIPv4Neighbor}};
            family inet-vpn {
                unicast;
            }
            family inet6-vpn {
                unicast;
            }
        }
```

```
                }
            }
            routing-options {
                    dynamic-tunnels {
                    dyn-tunnels {
                        source-address {{.Node.IPv4LoopbackAddr}};
                        udp;
                        destination-networks {{.Node.BGPIPv4Neighbor}}/32;
                    }
                }
            }
        }
    }
```

NOTE: You can define additional cRPD configuration hierarchies in the template. The values to be rendered from the annotations defined in the `node-annotations.yaml` must be defined as `{{.Node.`*`key`*`}}`. Any environment variables, such as variables defined in `values.yaml`, must be defined as `{{.Env.`*`variable_name`*`}}`.

Once the template is configured, you must copy the `jcnr-cni-custom-config.tmpl` file to the `Juniper_Cloud_Native_Router_`*`release_number`*`/helmchart/charts/jcnr-cni/files/` directory.

```
# cp Juniper_Cloud_Native_Router_release_number/helmchart/cRPD_examples/jcnr-cni-custom-
config.tmpl Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/
cp: overwrite 'Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/jcnr-
cni-custom-config.tmpl'? yes
#
```

Apply the node annotations to the cluster nodes using the command provided below:

```
# kubectl apply -f node-annotation.yaml
node/5d8s1-node1 configured
node/5d8s1-node2 configured
```

Follow the "JCNR installation instructions" on page 42 to deploy the cloud-native router components, including the cRPD. Once the installation completes, "access the cRPD shell" on page 52 and issue the `show configuration | display set` command in the `cli` mode to view the custom configuration you applied.

```
root@jcnr-01> show configuration
## Last commit: 2023-06-23 08:30:42 EDT by root
version 20230608.143922_builder.r1342735;
groups {
    base { /* OMITTED */ };
    custom {
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 110.1.1.2/32;
                    }
                    family iso {
                        address 49.0004.1000.0000.0001.00;
                    }
                }
            }
        }
        policy-options {
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then {
                    community add udp;
                }
            }
            community udp members encapsulation:0L:13;
        }
        routing-options {
            route-distinguisher-id 110.1.1.2;
            router-id 110.1.1.2;
            dynamic-tunnels {
                dyn-tunnels {
                    source-address 110.1.1.2;
                    udp;
                    destination-networks {
                        110.1.1.254/32;
                    }
                }
            }
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address 110.1.1.2;
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                    local-as 64512;
                    neighbor 110.1.1.254;
                }
            }
            isis {
                interface all;
                source-packet-routing {
                    srgb start-label 400000 index-range 4000;
                    node-segment {
                        ipv4-index 2000;
                        ipv6-index 3000;
                    }
                }
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
    }
    cni { /* OMITTED */ };
    internal { /* OMITTED */ };
}
apply-groups [ custom base internal ];
```

Troubleshooting

The cRPD pod continues to remain in `Pending` state if invalid configuration is rendered and applied via the go template. The rendered configuration is saved in `/etc/crpd` directory on the Cloud-Native Router host as `juniper.conf.master`. You can apply the rendered configuration manually to a running cRPD pod to validate the configuration and identify issues. For an AWS EKS deployment you can find the rendered config within the cRPD pod in the `/config` directory.

Install and Verify Juniper Cloud-Native Router

SUMMARY

The Juniper Cloud-Native Router (cloud-native router) uses the the JCNR-Controller (cRPD) to provide control plane capabilities and JCNR-CNI to provide a container network interface. Juniper Cloud-Native Router uses the DPDK-enabled vRouter to provide high-performance data plane capabilities and Syslog-NG to provide notification functions. This section explains how you can install these components of the Cloud-Native Router.

IN THIS SECTION

- Install Juniper Cloud-Native Router Using Helm Chart | **42**
- Verify Installation | **45**

Install Juniper Cloud-Native Router Using Helm Chart

Read this section to learn the steps required to load the cloud-native router image components into docker and install the cloud-native router components using Helm charts.

1. Review the "Before You Install" on page 21 section to ensure the cluster has all the required configuration.
2. Download the tarball, **Juniper_Cloud_Native_Router_*<release-number>*.tgz**, to the directory of your choice. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
3. Expand the file **Juniper_Cloud_Native_Router_*<release-number>*.tgz**.

```
tar xzvf Juniper_Cloud_Native_Router_<release-number>.tgz
```

4. Change directory to Juniper_Cloud_Native_Router_*<release-number>*.

```
cd Juniper_Cloud_Native_Router_<release-number>
```

NOTE: All remaining steps in the installation assume that your current working directory is now **Juniper_Cloud_Native_Router_*<release-number>***.

5. View the contents in the current directory.

```
ls
contrail-tools  helmchart  images  README.md  secrets
```

6. The Cloud-Native Router container images are required for deployment. Choose one of the following options:

   - Configure your cluster to deploy images from the Juniper Networks `enterprise-hub.juniper.net` repository. See "Configure Repository Credentials" on page 102 for instructions on how to configure repository credentials in the deployment Helm chart.

   - Configure your cluster to deploy images from the images tarball included in the downloaded Cloud-Native Router software package. See "Deploy Prepackaged Images" on page 103 for instructions on how to import images to the local container runtime.

7. Enter the root password for your host server into the **secrets/jcnr-secrets.yaml** file at the following line:

```
  root-password: <add your password in base64 format>
```

   You must enter the password in base64-encoded format. Encode your password as follows:

```
echo -n "password" | base64 -w0
```

   Copy the output of this command into **secrets/jcnr-secrets.yaml**.

8. Enter your Juniper Cloud-Native Router license into the **secrets/jcnr-secrets.yaml** file at the following line.

```
  crpd-license: |
    <add your license in base64 format>
```

You must enter your license in base64-encoded format. Encode your license as follows:

```
base64 -w0 licenseFile
```

Copy the output of this command into **secrets/jcnr-secrets.yaml**.

NOTE: You must obtain your license file from your account team and install it in the **jcnr-secrets.yaml** file as instructed above. Without the proper base64-encoded license key and root password in the **jcnr-secrets.yaml** file, the cRPD Pod does not enter Running state, but remains in CrashLoopBackOff state.

NOTE: Starting with Cloud-Native Router Release 23.2, the Cloud-Native Router license format has changed. Request a new license key from the JAL portal before deploying or upgrading to 23.2 or newer releases.

9. Apply **secrets/jcnr-secrets.yaml**.

```
kubectl apply -f secrets/jcnr-secrets.yaml
namespace/jcnr created
secret/jcnr-secrets created
```

10. Customize the helm chart for your deployment using the **helmchart/values.yaml** file.

    See, "Customize Cloud-Native Router Helm Chart" on page 22 for descriptions of the helm chart configurations.

11. If you are installing Cloud-Native Router on Amazon EKS, then update the `dpdkCommandAdditionalArgs` key in the **helmchart/charts/jcnr-vrouter/values.yaml** file and set `tx` and `rx` descriptors to 256, else skip this step.

    For example:

```
dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 256 --dpdk_rxd_sz 256"
```

12. Optionally, create cRPD pods with custom configuration.

    See, "Customize Cloud-Native Router Configuration using Node Annotations" on page 35 for creating and applying the cRPD customizations.

13. Deploy the Juniper Cloud-Native Router using the helm chart.

Navigate to the `helmchart` directory and run the following command:

```
helm install jcnr .
```

```
NAME: jcnr
LAST DEPLOYED: Fri Jun 23 06:04:33 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

14. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

Sample output:

```

NAME        NAMESPACE    REVISION        UPDATED
STATUS          CHART                      APP VERSION
jcnr        default      1               2023-06-23 06:04:33.144611017 -0400 EDT
deployed        jcnr-23.2.0                23.2.0
```

Verify Installation

This section enables you to confirm a successful Cloud-Native Router deployment.

1. Verify the state of the Cloud-Native Router pods by issuing the `kubectl get pods -A` command.

The output of the `kubectl` command shows all of the pods in the Kubernetes cluster in all namespaces. Successful deployment means that all pods are in the running state. In this example we have marked the Juniper Cloud-Native Router Pods in **bold**. For example:

```
kubectl get pods -A
```

```
NAMESPACE         NAME                                      READY   STATUS
RESTARTS         AGE
contrail-deploy   contrail-k8s-deployer-579cd5bc74-g27gs    1/1     Running
0                103s
contrail          contrail-vrouter-masters-lqjqk            3/3     Running
0                87s
jcnr              kube-crpd-worker-sts-0                    1/1     Running
0                103s
jcnr              syslog-ng-ds5qd                           1/1     Running
0                103s
kube-system       calico-kube-controllers-5f4fd8666-m78hk   1/1     Running   1 (3h13m
ago)    4h2m
kube-system       calico-node-28w98                         1/1     Running   3 (4d1h
ago)     86d
kube-system       coredns-54bf8d85c7-vkpgs                  1/1     Running
0                3h8m
kube-system       dns-autoscaler-7944dc7978-ws9fn           1/1     Running   3 (4d1h
ago)     86d
kube-system       kube-apiserver-ix-esx-06                  1/1     Running   4 (4d1h
ago)     86d
kube-system       kube-controller-manager-ix-esx-06         1/1     Running   8 (4d1h
ago)     86d
kube-system       kube-multus-ds-amd64-jl69w                1/1     Running   3 (4d1h
ago)     86d
kube-system       kube-proxy-qm5bl                          1/1     Running   3 (4d1h
ago)     86d
kube-system       kube-scheduler-ix-esx-06                  1/1     Running   9 (4d1h
ago)     86d
kube-system       nodelocaldns-bntfp                        1/1     Running   4 (4d1h
ago)     86d
```

2. Verify the Cloud-Native Router daemonsets by issuing the `kubectl get ds -A` command.

Use the `kubectl get ds -A` command to get a list of daemonsets. The Cloud-Native Router daemonsets are highlighted in bold text.

```
kubectl get ds -A
```

```
NAMESPACE     NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR             AGE
contrail      contrail-vrouter-masters    1         1         1       1            1
<none>                    90m
contrail      contrail-vrouter-nodes      0         0         0       0            0
<none>                    90m
jcnr          syslog-ng                   1         1         1       1            1
<none>                    90m
kube-system   calico-node                 1         1         1       1            1
kubernetes.io/os=linux    86d
kube-system   kube-multus-ds-amd64        1         1         1       1            1
kubernetes.io/arch=amd64  86d
kube-system   kube-proxy                  1         1         1       1            1
kubernetes.io/os=linux    86d
kube-system   nodelocaldns                1         1         1       1            1
kubernetes.io/os=linux    86d
```

3. Verify the Cloud-Native Router statefulsets by issuing the `kubectl get statefulsets -A` command.

   The command output provides the statefulsets.

```
kubectl get statefulsets -A
```

```
NAMESPACE   NAME                    READY   AGE
jcnr        kube-crpd-worker-sts    1/1     27m
```

4. Verify if the cRPD is licensed and has the appropriate configurations

   a. View the "Accessing the Cloud-Native Router Controller (cRPD) CLI" on page 52 section to access the cRPD CLI.

b. Once you have access the cRPD CLI, issue the `show system license` command in the cli mode to view the system licenses. For example:

```
root@jcnr-01:/# cli
root@jcnr-01> show system license
License usage:
                                 Licenses     Licenses    Licenses    Expiry
  Feature name                       used    installed      needed
  containerized-rpd-standard            1         1            0    2024-09-20 16:59:00 PDT

Licenses installed:
  License identifier: 85e5229f-0c64-0000-c10e4-a98c09ab34a1
  License SKU: S-CRPD-10-A1-PF-5
  License version: 1
  Order Type: commercial
  Software Serial Number: 1000098711000-iHpgf
  Customer ID: Juniper Networks Inc.
  License count: 15000
  Features:
    containerized-rpd-standard - Containerized routing protocol daemon with standard
features
      date-based, 2022-08-21 17:00:00 PDT - 2027-09-20 16:59:00 PDT
```

c. Issue the `show configuration | display set` command in the cli mode to view the cRPD default and custom configuration. The output will be based on the custom configuration and the Cloud-Native Router deployment mode.

```
root@jcnr-01# cli
root@jcnr-01> show configuration | display set
```

d. Type the `exit` command to exit from the pod shell.

5. Verify the vRouter interfaces configuration

a. View the "Accessing the vRouter CLI" on page 79 section to access the vRouter CLI.

b. Once you have accessed the vRouter CLI, issue the `vif --list` command to view the vRouter interfaces . The output will depend upon the Cloud-Native Router deployment mode and configuration. An example for L3 mode deployment, with one fabric interface configured, is provided below:

```
$ vif --list

Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload,
Mon=Interface is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag,
HbsL=HBS Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled

vif0/0      Socket: unix MTU: 1514
            Type:Agent HWaddr:00:00:5e:00:01:00
            Vrf:65535 Flags:L2 QOS:-1 Ref:3
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/1      PCI: 0000:5a:02.1 (Speed 10000, Duplex 1) NH: 6 MTU: 9000
            Type:Physical HWaddr:ba:9c:0f:ab:e2:c9 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:L3L2Vof QOS:0 Ref:12
            RX port   packets:66 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:5a:02.1  Status: UP  Driver: net_iavf
            RX packets:66  bytes:5116 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/2      PMD: eno3v1 NH: 9 MTU: 9000
            Type:Host HWaddr:ba:9c:0f:ab:e2:c9 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:65535 Flags:L3L2DProxyEr QOS:-1 Ref:13 TxXVif:1
```

```
              RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
              RX packets:0  bytes:0 errors:0
              TX packets:66  bytes:5116 errors:0
              Drops:0
              TX queue  packets:66 errors:0
              TX device packets:66  bytes:5116 errors:0
```

c. Type the `exit` command to exit the pod shell.

3

CHAPTER

Monitor

IN THIS CHAPTER

- Monitor Cloud-Native Router via CLI | 52
- Telemetry Capabilities of Cloud-Native Router | 59
- Logging and Notifications | 66

Monitor Cloud-Native Router via CLI

SUMMARY

This topic contains instructions to access the Cloud-Native Router controller (cRPD) CLI and run operational commands.

IN THIS SECTION

- Accessing the Cloud-Native Router Controller (cRPD) CLI | 52
- Example Show Commands | 54
- Example Clear Commands | 59

Accessing the Cloud-Native Router Controller (cRPD) CLI

You can access the command-line interface (CLI) of the cloud-native router controller by accessing the shell of the running cRPD container.

NOTE: The commands below are provided as an example. The cRPD pod name must be replaced from your environment. The command outputs may differ based on your environment.

List the K8s Pods Running in the Cluster

```
kubectl get pods -A
```

```
NAMESPACE        NAME                                        READY   STATUS      RESTARTS
AGE
contrail-deploy  contrail-k8s-deployer-7b5dd699b9-nd7xf      1/1     Running     0
41m
contrail         contrail-vrouter-masters-dfxgm              3/3     Running     0
41m
jcnr             kube-crpd-worker-ds-8tnf7                   1/1     Running     0
41m
jcnr             syslog-ng-54749b7b77-v24hq                  1/1     Running     0
41m
kube-system      calico-kube-controllers-57b9767bdb-5wbj6    1/1     Running     2 (92d ago)
129d
```

```
kube-system       calico-node-j4m5b                         1/1     Running     2 (92d ago)
129d
kube-system       coredns-8474476ff8-fpw78                  1/1     Running     2 (92d ago)
129d
kube-system       dns-autoscaler-7f76f4dd6-q5vdp            1/1     Running     2 (92d ago)
129d
kube-system       kube-apiserver-5a5s5-node2                1/1     Running     3 (92d ago)
129d
kube-system       kube-controller-manager-5a5s5-node2       1/1     Running     4 (92d ago)
129d
kube-system       kube-multus-ds-amd64-4zm5k                1/1     Running     2 (92d ago)
129d
kube-system       kube-proxy-l6xm8                          1/1     Running     2 (92d ago)
129d
kube-system       kube-scheduler-5a5s5-node2                1/1     Running     4 (92d ago)
129d
kube-system       nodelocaldns-6kwg5                        1/1     Running     2 (92d ago)
129d
```

Copy the name of the cRPD pod—`kube-crpd-worker-ds-8tnf7` in this example output . You will use the pod name to connect to the running container's shell.

Connect to the cRPD CLI

Issue the `kubectl exec` command to access the running container's shell:

```
kubectl exec -n <namespace> -it <pod name> --container <container name> -- bash
```

where *<namespace>* identifies the namespace in which the pod is running, *<pod name>* specificies the name of the pod and the *<container name>* specifies the name of the container (to be specified if the pod has more than one container).

The cRPD pod has only one running container. Here is an example command:

```
kubectl exec -n jcnr -it kube-crpd-worker-ds-8tnf7 -- bash
```

The result of the above command should appear similar to:

```
Defaulted container "kube-crpd-worker" out of: kube-crpd-worker, jcnr-crpd-config (init),
install-cni (init)

===>
```

```
           Containerized Routing Protocols Daemon (CRPD)
 Copyright (C) 2020-2022, Juniper Networks, Inc. All rights reserved.
                                                                      <===

root@jcnr-01:/#
```

At this point, you have connected to the shell of the cRPD. Just as with other Junos-based shells, you access the operational mode of the cloud-native router the same way as if you were connected to the console of a physical Junos OS device.

```
root@jcnr-01:/# cli
root@jcnr-cni>
```

Example Show Commands

Here are some example show commands you can execute:

```
show interfaces terse
Interface@link   Oper State     Addresses
__crpd-brd1      UNKNOWN        fe80::acbf:beff:fe8a:e046/64
cali1b684d67bd4@if3 UP             fe80::ecee:eeff:feee:eeee/64
cali34cf41e29bb@if3 UP             fe80::ecee:eeff:feee:eeee/64
docker0          DOWN           172.17.0.1/16
eno1             UP             10.102.70.146/24 fe80::a94:efff:fe79:dcae/64
eno2             UP
eno3             UP             10.1.1.1/24 fe80::a94:efff:fe79:dcac/64
eno3v1           UP
eno4             DOWN
enp0s20f0u1u6    UNKNOWN
ens2f0           DOWN
ens2f1           DOWN
erspan0@NONE     DOWN
eth0             UNKNOWN        169.254.143.126/32 fe80::b4db:eeff:fe78:9f43/64
gre0@NONE        UNKNOWN
gretap0@NONE     DOWN
ip6tnl0@NONE     UNKNOWN        fe80::74b6:2cff:fea7:d850/64
irb              DOWN
kube-ipvs0       DOWN           10.233.0.1/32 10.233.0.3/32 10.233.35.229/32
lo               UNKNOWN        127.0.0.1/8 ::1/128
lsi              UNKNOWN        fe80::cc59:6dff:fe9c:4db3/64
```

```
nodelocaldns     DOWN           169.254.25.10/32
sit0@NONE
UNKNOWN        ::169.254.143.126/96 ::10.233.91.64/96 ::172.17.0.1/96 ::10.102.70.146/96 ::10.1.1
.1/96 ::127.0.0.1/96
tunl0@NONE       UNKNOWN
vxlan.calico     UNKNOWN        10.233.91.64/32 fe80::64c6:34ff:fecd:3522/64
```

```
show configuration routing-instances
vswitch {
    instance-type virtual-switch;
    bridge-domains {
        bd100 {
            vlan-id 100;
        }
        bd200 {
            vlan-id 200;
        }
        bd300 {
            vlan-id 300;
        }
        bd700 {
            vlan-id 700;
            interface enp59s0f1v0;
        }
        bd701 {
            vlan-id 701;
        }
        bd702 {
            vlan-id 702;
        }
        bd703 {
            vlan-id 703;
        }
        bd704 {
            vlan-id 704;
        }
        bd705 {
            vlan-id 705;
        }
    }
    interface bond0;
}
```

```
show bridge ?
Possible completions:
mac-table       Show media access control table
statistics      Show bridge statistics information
```

```
show bridge mac-table ?
Possible completions:
  <[Enter]>            Execute this command
  count                Number of MAC address
  mac-address          MAC address in the format XX:XX:XX:XX:XX:XX
  vlan-id              Display MAC address learned on a specified VLAN or 'all-vlan'
  |                    Pipe through a command
```

```
show bridge mac-table
Routing Instance : default-domain:default-project:ip-fabric:__default__
Bridging domain VLAN id : 3002
MAC                  MAC                Logical
address              flags              interface

00:00:5E:00:53:01      D                 bond0
```

```
show bridge statistics ?
Possible completions:
  <[Enter]>            Execute this command
  vlan-id              Display statistics for a particular vlan (1..4094)
  |                    Pipe through a command
```

```
show bridge statistics
Bridge domain vlan-id: 100
   Local interface:  bond0
      Broadcast packets Tx  : 0           Rx  : 0
      Multicast packets Tx  : 0           Rx  : 0
      Unicast packets Tx    : 0           Rx  : 0
```

```
        Broadcast bytes Tx    : 0           Rx  : 0
        Multicast bytes Tx    : 0           Rx  : 0
        Unicast bytes Tx      : 0           Rx  : 0
        Flooded packets       : 0
        Flooded bytes         : 0
     Local interface: ens1f0v1
        Broadcast packets Tx  : 0           Rx  : 0
        Multicast packets Tx  : 0           Rx  : 0
        Unicast packets Tx    : 0           Rx  : 0
        Broadcast bytes Tx    : 0           Rx  : 0
        Multicast bytes Tx    : 0           Rx  : 0
        Unicast bytes Tx      : 0           Rx  : 0
        Flooded packets       : 0
        Flooded bytes         : 0
     Local interface: ens1f3v1
        Broadcast packets Tx  : 0           Rx  : 0
        Multicast packets Tx  : 0           Rx  : 0
        Unicast packets Tx    : 0           Rx  : 0
        Broadcast bytes Tx    : 0           Rx  : 0
        Multicast bytes Tx    : 0           Rx  : 0
        Unicast bytes Tx      : 0           Rx  : 0
        Flooded packets       : 0
```

```
show firewall filter filter1
Filter : filter1    vlan-id : 3001
 Term                 Packet
  t1                  0
```

```
show configuration firewall:firewall
family {
    bridge {
        filter filter1 {
            term t1 {
                from {
                    destination-mac-address 10:30:30:30:30:31;
                    source-mac-address 10:30:30:30:30:30;
                    ether-type oam;
                }
                then {
                    discard;
                }
            }
        }
    }
}
```

```
show route 172.68.20.2/32 table nad1.inet
nad1.inet.0: 11 destinations, 15 routes (11 active, 0 holddown, 0 hidden)
@ = Routing Use Only, # = Forwarding Use Only
+ = Active Route, - = Last Active, * = Both

172.68.20.2/32     @[BGP/170] 00:00:23, localpref 100, from 1.1.1.220
                      AS path: I, validation-state: unverified
                    >  via Tunnel Composite, UDP (src 1.1.1.35 dest 1.1.1.220), Push 48
                    [BGP/170] 00:13:18, localpref 100, from 1.1.24.24
                      AS path: I, validation-state: unverified
                    >  via Tunnel Composite, UDP (src 1.1.1.35 dest 1.1.24.24), Push 16
                   #[Multipath/255] 00:00:23, metric2 2
                       via Tunnel Composite, UDP (src 1.1.1.35 dest 1.1.1.220), Push 48
                    >  via Tunnel Composite, UDP (src 1.1.1.35 dest 1.1.24.24), Push 16
```

```
show interfaces routing enp216s0f0
Interface        State Addresses
enp216s0f0       Up    MPLS  enabled
                       ISO   enabled
                       INET  192.168.123.3
                       INET6 2001:192:168:123::3
                       INET6 fe80::42a6:b7ff:fe2c:a448
```

```
show dynamic-tunnels database
*- Signal Tunnels #- PFE-down
Table: inet.3
Destination-network: 1.1.1.220/32
Destination-network: 1.1.24.24/32
Tunnel to: 1.1.24.24/32
  Reference count: 4
  Next-hop type: UDP (forwarding-nexthop)
    Source address: 1.1.1.35
    Next hop: v6 mapped, tunnel-composite, 0x557917afc91c, nhid 0
```

```
        VPN Label: Push 16, Reference count: 2
        Ingress Route: [OSPF] 1.1.24.24/32, via metric 2
        Traffic Statistics: Packets 0, Bytes 0
        State: Up
    Aggregate Traffic Statistics:
```

Example Clear Commands

Here are some example clear commands:

```
clear bridge mac-table ?
Possible completions:
  <[Enter]>            Execute this command
  mac-address          Clear specific MAC address
  vlan-id              Clear mac-table for a specified vlan-id (1..4094)
  |                    Pipe through a command
```

```
clear bridge statistics ?
Possible completions:
  <[Enter]>            Execute this command
  vlan-id              Clear L2 interface statistics for a specified vlan-id (1..4094)
  |                    Pipe through a command
```

Telemetry Capabilities of Cloud-Native Router

IN THIS SECTION

- Cloud-Native Router Telemetry | 60

Read this topic to learn about the telemetry data available from Juniper Cloud-Native Router.

Cloud-Native Router Telemetry

Juniper Cloud-Native Router comes with telemetry capabilities that enable you to see performance metrics and telemetry data. The container **contrail-vrouter-telemetry-exporter** provides you this visibility. This container runs alongside the other vRouter containers in the **contrail-vrouter-masters** pod.

The telemetry exporter periodically queries the Introspect on the vRouter-agent for statistics and reports metrics information in response to the Prometheus scrape requests. You can directly view the telemetry data by using the following URL: **http://*host server IP address*:8070**. The following table shows a sample output.

NOTE: We've grouped the output shown in the following table. The cloud-native router does not group or sort the output on live systems.

Table 6: Sample Telemetry Output

Group	Sample Output

Memory usage per vRouter	`# TYPE virtual_router_system_memory_cached_bytes gauge`
	`# HELP virtual_router_system_memory_cached_bytes Virtual router system memory cached`
	`virtual_router_system_memory_cached_bytes{vrouter_name="jcnr.example.com"} 2635970448`
	`# TYPE virtual_router_system_memory_buffers gauge`
	`# HELP virtual_router_system_memory_buffers Virtual router system memory buffer`
	`virtual_router_system_memory_buffers{vrouter_name="jcnr.example.com"} 32689`
	`# TYPE virtual_router_system_memory_bytes gauge`
	`# HELP virtual_router_system_memory_bytes Virtual router total system memory`
	`virtual_router_system_memory_bytes{vrouter_name="jcnr.example.com"} 2635970448`
	`# TYPE virtual_router_system_memory_free_bytes gauge`
	`# HELP virtual_router_system_memory_free_bytes Virtual router system memory free`
	`virtual_router_system_memory_free_bytes{vrouter_name="jcnr.example.com"} 2635969296`
	`# TYPE virtual_router_system_memory_used_bytes gauge`
	`# HELP virtual_router_system_memory_used_bytes Virtual router system memory used`
	`virtual_router_system_memory_used_bytes{vrouter_name="jcnr.example.com"} 32689`
	`# TYPE virtual_router_virtual_memory_kilobytes gauge`
	`# HELP virtual_router_virtual_memory_kilobytes Virtual router virtual memory`
	`virtual_router_virtual_memory_kilobytes{vrouter_name="jcnr.example.com"} 0`
	`# TYPE virtual_router_resident_memory_kilobytes gauge`
	`# HELP virtual_router_resident_memory_kilobytes Virtual router resident memory`
	`virtual_router_resident_memory_kilobytes{vrouter_name="jcnr.example.com"} 32689`
	`# TYPE virtual_router_peak_virtual_memory_bytes gauge`
	`# HELP virtual_router_peak_virtual_memory_bytes Virtual router peak virtual memory`
	`virtual_router_peak_virtual_memory_bytes{vrouter_name="jcnr.example.com"} 2894328001`

Table 6: Sample Telemetry Output *(Continued)*

Group	Sample Output

Packet count per interface	
	```
	# TYPE virtual_router_phys_if_input_packets_total counter
	# HELP virtual_router_phys_if_input_packets_total Total packets received by physical
	interface
	virtual_router_phys_if_input_packets_total{vrouter_name="jcnr.example.com",interface_na
	me="bond0"} 1483
	# TYPE virtual_router_phys_if_output_packets_total counter
	# HELP virtual_router_phys_if_output_packets_total Total packets sent by physical
	interface
	virtual_router_phys_if_output_packets_total{vrouter_name="jcnr.example.com",interface_n
	ame="bond0"} 32969
	# TYPE virtual_router_phys_if_input_bytes_total counter
	# HELP virtual_router_phys_if_input_bytes_total Total bytes received by physical
	interface
	virtual_router_phys_if_input_bytes_total{interface_name="bond0",vrouter_name="jcnr.exam
	ple.com"} 125558
	# TYPE virtual_router_phys_if_output_bytes_total counter
	# HELP virtual_router_phys_if_output_bytes_total Total bytes sent by physical interface
	virtual_router_phys_if_output_bytes_total{vrouter_name="jcnr.example.com",interface_nam
	e="bond0"} 4597076
	virtual_router_phys_if_input_bytes_total{vrouter_name="jcnr.example.com",interface_name
	="bond0"} 228300499320
	virtual_router_phys_if_output_bytes_total{interface_name="bond0",vrouter_name="jcnr.exa
	mple.com"} 228297889634
	virtual_router_phys_if_input_packets_total{interface_name="bond0",vrouter_name="jcnr.ex
	ample.com"} 1585421179
	virtual_router_phys_if_output_packets_total{vrouter_name="jcnr.example.com",interface_n
	ame="bond0"} 1585402623
	virtual_router_phys_if_output_packets_total{interface_name="bond0",vrouter_name="jcnr.e
	xample.com"} 1585403344
	```

Table 6: Sample Telemetry Output *(Continued)*

Group	Sample Output

CPU usage per vRouter	`# TYPE virtual_router_cpu_1min_load_avg gauge`
	`# HELP virtual_router_cpu_1min_load_avg Virtual router CPU 1 minute load average`
	`virtual_router_cpu_1min_load_avg{vrouter_name="jcnr.example.com"} 0.11625`
	`# TYPE virtual_router_cpu_5min_load_avg gauge`
	`# HELP virtual_router_cpu_5min_load_avg Virtual router CPU 5 minute load average`
	`virtual_router_cpu_5min_load_avg{vrouter_name="jcnr.example.com"} 0.109687`
	`# TYPE virtual_router_cpu_15min_load_avg gauge`
	`# HELP virtual_router_cpu_15min_load_avg Virtual router CPU 15 minute load average`
	`virtual_router_cpu_15min_load_avg{vrouter_name="jcnr.example.com"} 0.110156`

Drop packet count per vRouter	`# TYPE virtual_router_dropped_packets_total counter`
	`# HELP virtual_router_dropped_packets_total Total packets dropped`
	`virtual_router_dropped_packets_total{vrouter_name="jcnr.example.com"} 35850`

Table 6: Sample Telemetry Output ***(Continued)***

Group	Sample Output

Packet count per interface per VLAN	`# TYPE virtual_router_interface_vlan_multicast_input_packets_total counter`
	`# HELP virtual_router_interface_vlan_multicast_input_packets_total Total number of multicast packets received on interface VLAN`
	`virtual_router_interface_vlan_multicast_input_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_broadcast_output_packets_total counter`
	`# HELP virtual_router_interface_vlan_broadcast_output_packets_total Total number of broadcast packets sent on interface VLAN`
	`virtual_router_interface_vlan_broadcast_output_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_broadcast_input_packets_total counter`
	`# HELP virtual_router_interface_vlan_broadcast_input_packets_total Total number of broadcast packets received on interface VLAN`
	`virtual_router_interface_vlan_broadcast_input_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_multicast_output_packets_total counter`
	`# HELP virtual_router_interface_vlan_multicast_output_packets_total Total number of multicast packets sent on interface VLAN`
	`virtual_router_interface_vlan_multicast_output_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_unicast_input_packets_total counter`
	`# HELP virtual_router_interface_vlan_unicast_input_packets_total Total number of unicast packets received on interface VLAN`
	`virtual_router_interface_vlan_unicast_input_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_flooded_output_bytes_total counter`
	`# HELP virtual_router_interface_vlan_flooded_output_bytes_total Total number of output bytes flooded to interface VLAN`
	`virtual_router_interface_vlan_flooded_output_bytes_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_multicast_output_bytes_total counter`
	`# HELP virtual_router_interface_vlan_multicast_output_bytes_total Total number of multicast bytes sent on interface VLAN`
	`virtual_router_interface_vlan_multicast_output_bytes_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_unicast_output_packets_total counter`
	`# HELP virtual_router_interface_vlan_unicast_output_packets_total Total number of unicast packets sent on interface VLAN`
	`virtual_router_interface_vlan_unicast_output_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_broadcast_input_bytes_total counter`

Table 6: Sample Telemetry Output *(Continued)*

Group	Sample Output

	`# HELP virtual_router_interface_vlan_broadcast_input_bytes_total Total number of broadcast bytes received on interface VLAN`
	`virtual_router_interface_vlan_broadcast_input_bytes_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_multicast_input_bytes_total counter`
	`# HELP virtual_router_interface_vlan_multicast_input_bytes_total Total number of multicast bytes received on interface VLAN`
	`virtual_router_interface_vlan_multicast_input_bytes_total{vlan_id="100",interface_id="1"} 0`
	`# TYPE virtual_router_interface_vlan_unicast_input_bytes_total counter`
	`# HELP virtual_router_interface_vlan_unicast_input_bytes_total Total number of unicast bytes received on interface VLAN`
	`virtual_router_interface_vlan_unicast_input_bytes_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_flooded_output_packets_total counter`
	`# HELP virtual_router_interface_vlan_flooded_output_packets_total Total number of output packets flooded to interface VLAN`
	`virtual_router_interface_vlan_flooded_output_packets_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_broadcast_output_bytes_total counter`
	`# HELP virtual_router_interface_vlan_broadcast_output_bytes_total Total number of broadcast bytes sent on interface VLAN`
	`virtual_router_interface_vlan_broadcast_output_bytes_total{interface_id="1",vlan_id="100"} 0`
	`# TYPE virtual_router_interface_vlan_unicast_output_bytes_total counter`
	`# HELP virtual_router_interface_vlan_unicast_output_bytes_total Total number of unicast bytes sent on interface VLAN`
	`virtual_router_interface_vlan_unicast_output_bytes_total{interface_id="1",vlan_id="100"} 0`
	`...`

Prometheus is an open-source systems monitoring and alerting toolkit. You can use Prometheus to retrieve telemetry data from the cloud-native router host servers and view that data in the HTTP format. A sample of Prometheus configuration looks like this:

```
- job_name: "prometheus-JCNR-1a2b3c"

# metrics_path defaults to '/metrics'
# scheme defaults to 'http'.
```

```
static_configs:
- targets: ["<host-server-IP>:8070"]
```

Logging and Notifications

IN THIS SECTION

- Logging | 66
- Notifications | 67

Read this topic to learn about logging and notification functions in Juniper Cloud-Native Router. We discuss the location of log files, what you can log, and various log levels. You can also learn about the available notifications and how the notifications are implemented in the cloud-native router.

Logging

The Juniper Cloud-Native Router pods and containers use syslog as their logging mechanism. You can determine the location of the log files at the deployment time by retaining or changing the value of the **log_path** key in the **values.yaml** file. By default, the location of the log files is **/var/log/jcnr**. The system stores log files from all the cloud-native router pods and containers in the **log_path** directory.

In addition, a syslog-ng pod stores event notification data in JSON format on the host server. The syslog-ng pod stores the JSON-formatted notifications in the directory specified by the **syslog_notifications** key in the **values.yaml** file. By default, the file location is **/var/log/jcnr** and the filename is **jcnr_notifications.json**. You can change the location and filename by changing the value of the **syslog_notifications** key before the cloud-native router deployment.

When you use the default file locations, the **/var/log/jcnr** directory displays the following files:

```
[root@jcnr-01 jcnr]# ls
action.log                    contrail-vrouter-dpdk-init.log  filter
l2cos.log       __policy_names_rpdc__
contrail-vrouter-agent.log    contrail-vrouter-dpdk.log       filter.log
license         mgd-api
```

```
__policy_names_rpdn__        cos                              jcnr-cni.log
messages       mosquitto
vrouter-kernel-init.log      cscript.log                      jcnr_notifications.json
messages.0.gz  na-grpcd
```

NOTE: The host server must manage the log rotation for the **contrail-vrouter-dpdk.log** and the **jcnr-cni.log** files.

Notifications

The syslog-ng pod continuously monitors the preceding log files for notification events such as interface up, interface down, interface add, and so on. When these events appear in a log file, syslog-ng converts the log events into notification events and stores the events in JSON format within the **syslog_notifications** file configured in the **values.yaml** file.

Here is a sample of syslog-ng notifications:

Table 7: Supported Notifications

Notification	Source Pod
License Near Expiry	cRPD
License Expired	cRPD
License Invalid	cRPD
License OK	cRPD
License Grace Period	cRPD
License Not Present	cRPD
Cloud-Native Router Init Success	Deployer
Cloud-Native Router Init Failure	Deployer
Cloud-Native Router Graceful Shutdown Request	Deployer

Table 7: Supported Notifications *(Continued)*

Notification	Source Pod
Cloud-Native Router Graceful Shutdown Complete	Deployer
Cloud-Native Router Graceful Shutdown Failure	Deployer
Cloud-Native Router Restart	Deployer
Cloud-Native Router Upgrade Success	Deployer
Cloud-Native Router Upgrade Failure	Deployer
Upstream Fabric Bond Member Link Up	vRouter
Upstream Fabric Bond Member Link Down	vRouter
Upstream Fabric Bond Link Up	vRouter
Upstream Fabric Bond Link Down	vRouter
Upstream Fabric Bond Link Switchover	vRouter
Downstream Fabric Link Up	vRouter
Downstream Fabric Link Down	vRouter
Appliance Link Up	vRouter
Appliance Link Down	vRouter
Any Cloud-Native Router Application Critical Errors	vRouter
Any Cloud-Native Router Application Warnings	vRouter
Any Cloud-Native Router Application Info	vRouter
Cloud-Native Router Rate Limits Reached	vRouter

Table 7: Supported Notifications *(Continued)*

Notification	Source Pod
Cloud-Native Router MAC Table Limit Reached	vRouter
Cloud-Native Router CLI Start	cRPD or vRouter-Agent
Cloud-Native Router CLI Stop	cRPD or vRouter-Agent
Cloud-Native Router Kernel App Interface Up	vRouter
Cloud-Native Router Kernel App Interface Down	vRouter
Cloud-Native Router Virtio User Interface Up	vRouter
Cloud-Native Router Virtio User Interface Down	vRouter

4

CHAPTER

Manage

IN THIS CHAPTER

- Manage Juniper Cloud-Native Router | 71

Manage Juniper Cloud-Native Router

IN THIS SECTION

- Upgrading JCNR | 71
- Downgrading JCNR | 71
- Uninstalling JCNR | 71

This topic provides high-level information about the available upgrade, downgrade and uninstall options for JCNR.

Upgrading JCNR

Currently, there is no procedure for upgrading to the Cloud-Native Router release 23.2. To change from a current version to 23.2, you must uninstall the current version and install the newer version.

Downgrading JCNR

Currently, there is no procedure for downgrading to an older version. To change from a current version to an older version, you must uninstall the current version and install an older version.

Uninstalling JCNR

Cloud-Native Router can be uninstalled by using the following command:

```
helm uninstall jcnr
```

Uninstalling Cloud-Native Router restores interfaces to their original state by unbinding from DPDK and binding back to the original driver. It also delete contents of Cloud-Native Router directories, deletes cRPD created interfaces and removes any Kubernetes objects created for JCNR.

NOTE: The **jcnr** namespace is not deleted as a part of the helm uninstallation and must be deleted manually.

After the triggering of helm uninstall command, please wait for all Kubernetes resources to be fully deleted before attempting a re-installation. Premature re-installation can lead to installation stalls and may require manual steps for recovery. The recovery steps are provided below:

```
helm uninstall jcnr --no-hooks
kubectl delete <ds/name>
kubectl delete <job/jobname>
kubectl delete ns jcnrops
```

5

CHAPTER

Troubleshoot

IN THIS CHAPTER

- Troubleshoot Deployment Issues | **74**
- Troubleshoot via the vRouter CLI | **78**
- Troubleshoot via Introspect | **91**

Troubleshoot Deployment Issues

SUMMARY

This topic provides information about how to troubleshoot deployment issues using Kubernetes commands and how to view the cloud-native router configuration files.

IN THIS SECTION

- Troubleshoot Deployment Issues | 74

Troubleshoot Deployment Issues

IN THIS SECTION

- Verify Cloud-Native Router Controller Configuration | 76
- View Log Files | 77
- Uninstallation Issues | 78

This topic provides information on some of the issues that might be seen during deployment of the cloud-native router components and provides a number of Kubernetes (K8s) and shell commands that you run on the host server to help determine the cause of deployment issues.

Table 8: Investigate Deployment Issues

Potential issue	What to check	Related Commands
Image not found	Check if the images are uploaded to the local docker using the command `docker images`. If not, then the registry configured in `values.yaml` should be accessible. Ensure image tags are correct.	• `kubectl -n jcnr describe pod <crpd-pod-name>`

Table 8: Investigate Deployment Issues *(Continued)*

Potential issue	What to check	Related Commands

Initialization errors	Check if jcnr-secrets is loaded and has a valid license key	`[root@jcnr-01]# kubectl get secrets -n jcnr`
		`NAME`
		`TYPE`
		`    DATA   AGE`
		`crpd-token-zp8kc`
		`kubernetes.io/service-account-token   3      29d`
		`default-token-zn6p9`
		`kubernetes.io/service-account-token   3      29d`
		`jcnr-secrets`
		`Opaque`
		`    2      29d`
		
		Confirm that root password and license key are present in `/var/run/jcnr/juniper.conf`

Table 8: Investigate Deployment Issues *(Continued)*

Potential issue	What to check	Related Commands

cRPD Pod in CrashLoopBackOff state	• Check if startup/liveness probe is failing or vrouter pod not running
	• rpd-vrouter-agent gRPC connection not UP
	• Composed configuration is invalid or config template is invalid
		• `kubectl get pods -A`
		`kubectl -n jcnr describe pod <crpd-pod-name>`
		`tail -f /var/log/jcnr/jcnr-cni.log`
		`tail -f /var/log/jcnr/jcnr_notifications.json`
		• See "Monitor Cloud-Native Router via CLI" on page 52 to enter the cRPD CLI and run the following command:
		`show krt state channel vrouter`
		• `cat /var/run/jcnr/juniper.conf`

vRouter Pod in CrashLoopBackOff state	Check the contail-k8s-deployer pod for errors	`kubectl logs contrail-k8s-deployer-<pod-hash> -n contrail-deploy`

Verify Cloud-Native Router Controller Configuration

The cloud-native router deployment process creates a configuration file for the cloud-native router controller (cRPD) as a result of entries in the **values.yaml** file for L2 mode and custom configuration via node annotations in L3 mode. You can view this configuration file to see the details of the cRPD configuration. To view the cRPD configuration, navigate to the `/var/run/jcnr` folder to access the configuration file details and view the contents of the configuration file.

```
[root@jcnr-01]# ls
cni  config  containers  envars  juniper.conf  reboot-canary
[root@jcnr-01]# cat juniper.conf
```

The cRPD configuration may be customized using "node annotations" on page 35. The cRPD pod will stay in `pending` state if the applied configuration is invalid.

You can view the rendered custom configuration in the `/etc/crpd/` directory.

```
[root@jcnr-01]# cat /etc/crpd/juniper.conf.master
```

In an AWS EKS deployment you can review the rendered custom configuration by "accessing the cRPD shell" on page 52 and reviewing the contents of the `/config` directory.

View Log Files

You can view the jcnr log files in the default log_path directory, **/var/log/jcnr/**. You can change the location of the log files by changing the value of the **log_path:** or **syslog_notifications:** keys in the **values.yaml** file prior to deployment.

Navigate to the following path and issue the `ls` command to list the log files for each of the cloud-native router components.

```
cd /var/log/jcnr/
```

```
[root@jcnr-01 jcnr]# ls
action.log                    contrail-vrouter-dpdk-init.log  filter
l2cos.log      __policy_names_rpdc__
contrail-vrouter-agent.log    contrail-vrouter-dpdk.log       filter.log
license        mgd-api
__policy_names_rpdn__         cos                             jcnr-cni.log
messages       mosquitto
vrouter-kernel-init.log       cscript.log                     jcnr_notifications.json
messages.0.gz  na-grpcd
```

Uninstallation Issues

After the triggering of helm uninstall command, please wait for all Kubernetes resources to be fully deleted before attempting a re-installation. Premature re-installation can lead to installation stalls and may require manual steps for recovery. The recovery steps are provided below:

```
helm uninstall jcnr -no-hooks
kubectl delete <ds/name>
kubectl delete <job/jobname>
kubectl delete ns jcnrops
```

Troubleshoot via the vRouter CLI

IN THIS SECTION

- Accessing the vRouter CLI | **79**
- Troubleshooting via the vRouter CLI | **80**

Read this topic to learn about the various troubleshooting commands available in the vRouter CLI. The following commands are covered in this topic:

- "Accessing the vRouter CLI" on page 79
- "Verify vRouter Interfaces via the vif Command" on page 80
- "View the running configuration of the vRouter" on page 82
- "View L2 Configuration and Statistics via the purel2cli Command" on page 83
- "The dropstats Command" on page 85
- "The dpdkinfo Command" on page 86
- "The rt and nh Commands" on page 89
- "The flow Command" on page 90

Accessing the vRouter CLI

You can access the command-line interface (CLI) of the vRouter by accessing the shell of the running vRouter-agent container.

NOTE: The commands below are provided as an example. The vRouter pod name must be replaced from your environment. The command outputs may differ based on your environment.

List the K8s Pods running on the cluster

```
kubectl get pods -A
```

```
NAMESPACE         NAME                                       READY    STATUS      RESTARTS
AGE
contrail-deploy   contrail-k8s-deployer-7b5dd699b9-nd7xf     1/1      Running     0
41m
contrail          contrail-vrouter-masters-dfxgm             3/3      Running     0
41m
jcnr              kube-crpd-worker-ds-8tnf7                  1/1      Running     0
41m
jcnr              syslog-ng-54749b7b77-v24hq                 1/1      Running     0
41m
kube-system       calico-kube-controllers-57b9767bdb-5wbj6   1/1      Running     2 (92d ago)
129d
kube-system       calico-node-j4m5b                          1/1      Running     2 (92d ago)
129d
kube-system       coredns-8474476ff8-fpw78                   1/1      Running     2 (92d ago)
129d
kube-system       dns-autoscaler-7f76f4dd6-q5vdp             1/1      Running     2 (92d ago)
129d
kube-system       kube-apiserver-5a5s5-node2                 1/1      Running     3 (92d ago)
129d
kube-system       kube-controller-manager-5a5s5-node2        1/1      Running     4 (92d ago)
129d
kube-system       kube-multus-ds-amd64-4zm5k                 1/1      Running     2 (92d ago)
129d
kube-system       kube-proxy-l6xm8                           1/1      Running     2 (92d ago)
129d
kube-system       kube-scheduler-5a5s5-node2                 1/1      Running     4 (92d ago)
```

```
129d
kube-system       nodelocaldns-6kwg5                        1/1     Running     2 (92d ago)
129d
```

Copy the name of the vRouter pod—`contrail-vrouter-masters-dfxgm` in this example output . You will use the pod name to connect to the running container's shell.

Connect to the vRouter CLI

Issue the `kubectl exec` command to access the running container's shell:

```
kubectl exec -n <namespace> -it <pod name> --container <container name> -- bash
```

where *<namespace>* identifies the namespace in which the pod is running, *<pod name>* specificies the name of the pod and the *<container name>* specifies the name of the container (to be specified if the pod has more than one container).

The vRouter pod has three containers. When the container name is not specified, the command will default to the vrouter-agent container shell. Here is an example:

```
[root@jcnr-01]# kubectl exec -n contrail -it contrail-vrouter-masters-dfxgm -- bash
Defaulted container "contrail-vrouter-agent" out of: contrail-vrouter-agent, contrail-vrouter-
agent-dpdk,
contrail-vrouter-telemetry-exporter, contrail-init (init), contrail-vrouter-kernel-init-dpdk
(init)
[root@jcnr-01 /]#
```

At this point, you have connected to the vRouter's CLI.

Troubleshooting via the vRouter CLI

You can run commands in the CLI to learn about the state of the vRouter.

Verify vRouter Interfaces via the `vif` Command

The command shown below allows you to see which interfaces are present on the vRouter:

```
vif --list
Vrouter Operation Mode: PureL2
Vrouter Interface Table
```

```
Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload, Mon=Interface
is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag, HbsL=HBS
Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled

vif0/0      Socket: unix
            Type:Agent HWaddr:00:00:5e:00:01:00
            Vrf:65535 Flags:L2 QOS:-1 Ref:3
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:0  bytes:0 errors:0
            TX packets:11  bytes:4169 errors:0
            Drops:0

vif0/1      PCI: 0000:00:00.0 (Speed 25000, Duplex 1)
            Type:Physical HWaddr:46:37:1f:de:df:bc
            Vrf:65535 Flags:L2Vof QOS:-1 Ref:8
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: eth_bond_bond0  Status: UP  Driver: net_bonding
            Slave Interface(0): 0000:3b:02.0  Status: UP  Driver: net_iavf
            Slave Interface(1): 0000:3b:02.1  Status: UP  Driver: net_iavf
            Vlan Mode: Trunk  Vlan: 100 200 300 700-705
            RX packets:0  bytes:0 errors:0
            TX packets:378  bytes:81438 errors:0
            Drops:0

vif0/2      PCI: 0000:3b:0a.0 (Speed 25000, Duplex 1)
            Type:Workload HWaddr:ba:69:c0:b7:1f:ba
            Vrf:0 Flags:L2Vof QOS:-1 Ref:7
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:3b:0a.0  Status: UP  Driver: net_iavf
            Vlan Mode: Access  Vlan Id: 700  OVlan Id: 700
            RX packets:378  bytes:81438 errors:2
            TX packets:0  bytes:0 errors:0
            Drops:391
```

View the running configuration of the vRouter

To see the status of the vRouter, enter the following command in the vRouter CLI:

```
[root@jcnr-01 /]# ps -eaf | grep vrouter-dpdk
root         116      90 99 Mar30 ?        118-08:05:37 /contrail-vrouter-dpdk --no-daemon --
socket-mem=1024 1024
--allow=0000:5a:02.0 --
vdev=eth_bond_bond0,mode=1,socket_id=0,mac=3a:1a:b7:86:1c:4f,primary=0000:5a:02.0,
slave=0000:5a:02.0 --l2_table_size=10240 --yield_option 0 --ddp --l2_mode
root     1134749 1134365  0 16:41 pts/0    00:00:00 grep --color=auto vrouter-dpdk
```

The output contains several elements.

Table 9: vRouter Status Attributes

Flag	Meaning
`--l2_mode`	The vRouter is running in L2 mode.
`--l2_table_size`	The current number of entries in the MAC table. The default size is 10240 entries.
`--allow=<PCI Id>`	The PCI ID of fabric and fabric workload interfaces. More than one ID can appear in the output. These IDs serve as an allowlist.
`--ddp`	Enable Intel DDP support. We enable DDP by default in the **values.yaml** file in the vRouter. **NOTE**: The Intel XL710 NIC does not support DDP.

View L2 Configuration and Statistics via the `purel2cli` Command

The `purel2cli` command is a useful utility to view the Cloud-Native Router L2 configuration and statistics. Start by using the **`purel2cli --help`** command.

```
[root@jcnr-01 /]# purel2cli --help
Usage: purel2cli [--mac show]
           [--vlan show]
           [--vlan get <VLAN_ID>]
           [--acl show <VLAN_ID>]
           [--acl reset-counters <VLAN_ID>]
           [--l2stats get <VIF_ID> <VLAN_ID>]
           [--clear VLAN_ID]
           [--qos classifier/re-write/scheduler <NAME>]
           [--qos cla/rw/sch <NAME>]
           [--nolocal show]
           [--nolocal get <VLAN_ID>]
           [--sock-dir <sock dir>]
           [--help]
```

The **`purel2cli --mac show`** command shows the MAC addresses that the vRouter has dynamically learned.

```
purel2cli --mac show
==================================================
||  MAC              vlan      port      hit_count||
==================================================
00:01:01:01:01:03  1221      2          1101892
00:01:01:01:01:02  1221      2          1101819
00:01:01:01:01:04  1221      2          1101863
00:01:01:01:01:01  1221      2          1101879
5a:4c:4c:75:90:fe  1250      5          12
Total Mac entries 5
```

The **purel2cli --vlan show** command shows the VLANs and associated ports.

```
purel2cli --vlan show
VLAN      PORT
===============
1201      1,2,3,4,
1202      1,2,3,4,
1203      1,2,3,4,
1204      1,2,3,4,
1205      1,2,3,4,
```

You can also issue the **`purel2cli --vlan get`** command to get more details about the VLAN.

```
purel2cli --vlan get <vlan-id>
```

Issue the **`purel2cli --l2stats`** command to view L2 statistics. For example:

```
purel2cli -- l2stats get <virtual_interface_ID> <VLAN_ID>
```

```
purel2cli --l2stats get 2 1221Vlan id count: 1
--------------------------------------------------------------------------------
Statistics for vif 2 vlan 1221
--------------------------------------------------------------------------------
                Rx Pkts           Rx Bytes          Tx Pkts            Tx Bytes
Unicast       245344824        48152682842           835552          1667761792
Broadcast             0                  0                0                   0
Multicast             0                  0                0                   0
Flood                 0                  0                0                   0
--------------------------------------------------------------------------------
```

```
purel2cli --clear '*'
```

```
purel2cli --clear 100
```

Table 10: `purel2cli` Command Options for L2 Statistics

Sample Command	Function
`purel2cli --l2stats get '*' '*'`	Get statistics for all virtual interfaces (vif) and all VLAN IDs.
`purel2cli --l2stats get '*' 100`	Get statistics for all vif that are part of VLAN 100

Table 10: `purel2cli` Command Options for L2 Statistics *(Continued)*

Sample Command	Function
`purel2cli --l2stats get 1 '*'`	Get statistics for all VLANs for which interface 1 is a member
`purel2cli --l2stats get 1 100`	Get statistics for interface 1 and VLAN 100

The command shows the VLAN to port mapping in the vRouter.You can use the command to see the bridge domain table entry for a specific VLAN: There are several variations of the command that allow you to display and filter L2 statistics in the vRouter. The base form of the command is: . The table below shows the available command options and what they do. It also provides a sample output using one of the options:The following command is an example of the L2 statistics for interface 2 and VLAN 1221:You can clear the statistics from the vRouter with the purel2cli command in the form: . Clears all statistics from all VLANs in the vRouter. Clears all statistics for VLAN id 100.

The `dropstats` Command

The vRouter tracks the packets that it drops and includes the reason for dropping them. The table below shows the common reasons for vRouter to drop a packet. When you execute the **dropstats** command, the vRouter does not show a counter if the count for that counter is 0.

Table 11: Dropstats Counters

Counter Name	Meaning
`L2 bd table drop`	No interfaces in bridge domain
`L2 untag pkt drop`	Untagged packet arrives on trunk or sub-interface
`L2 Invalid Vlan`	Packet VLAN does not match interface VLAN
`L2 Mac Table Full`	No more entries available in the MAC table
`L2 ACL drop`	Packet matched firewall filter (ACL) drop rule
`L2 Src Mac lookup fail`	Unable to match (or learn) the source MAC address

Example output from the **dropstats** command looks like:

```
dropstats
L2 bd table Drop              43
L2 untag pkt drop             716
L2 Invalid Vlan               7288253
Rate limit exceeded           673179706
L2 Mac Table Full             41398787
L2 ACL drop                   8937037
L2 Src Mac lookup fail        247046
```

The `dpdkinfo` Command

The **dpdkinfo** command provides insight into the status and statistics of DPDK. The **dpdkinfo** command has many options. The following sections describe the available options and the example output from the **dpdkinfo** command. You can run the **dpdkinfo** command only from within the vRouter-agent CLI.

```
dpdkinfo --help
Usage: dpdkinfo [--help]
                 --version|-v                                                   Show DPDK
Version
                 --bond|-b                                                      Show Master/
Slave bond information
                 --lacp|-l     <all/conf>                                       Show LACP
information from DPDK
                 --mempool|-m  <all/<mempool-name>>                             Show Mempool
information
                 --stats|-n    <vif index value>                                Show Stats
information
                 --xstats|-x   <vif index value>                                Show Extended
Stats information
                 --lcore|-c                                                     Show Lcore
information
                 --app|-a                                                       Show App
information
                 --ddp|-d      <list> <list-flow>                        Show DDP information
for X710 NIC
                 --rx_vlan|-z  <value>                                          Show VLan
information
```

```
       Optional: --buffsz      <value>                                       Send output
buffer size (less than 1000Mb)
```

The command **`dpdkinfo -c`** shows the Lcores assigned to DPDK VF fabric interfaces and the queue ID for each interface.

```
dpdkinfo -c
No. of forwarding lcores: 4

Lcore 10:
    Interface: 0000:18:01.1       Queue ID: 0
    Interface: 0000:18:0d.1       Queue ID: 0
    Interface: 0000:86:00.0       Queue ID: 0

Lcore 11:
    Interface: 0000:18:01.1       Queue ID: 1
    Interface: 0000:18:0d.1       Queue ID: 1
    Interface: 0000:86:00.0       Queue ID: 1

Lcore 12:
    Interface: 0000:18:01.1       Queue ID: 2
    Interface: 0000:18:0d.1       Queue ID: 2
    Interface: 0000:86:00.0       Queue ID: 2

Lcore 13:
    Interface: 0000:18:01.1       Queue ID: 3
    Interface: 0000:18:0d.1       Queue ID: 3
    Interface: 0000:86:00.0       Queue ID: 3
```

The command **`dpdkinfo -m all`** shows all of the memory pool information.

```
dpdkinfo -m all
---------------------------------------------------
Name            Size    Used    Available
---------------------------------------------------
rss_mempool             16384    1549    14835
frag_direct_mempool     4096    0    4096
frag_indirect_mempool    4096    0    4096
packet_mbuf_pool        8192    2    8190
```

The command **`dpdkinfo -n 3`** displays statistical information for a specific interface.

```
dpdkinfo -n 3
Interface Info(0000:18:0d.1):
RX Device Packets:6710, Bytes:1367533, Errors:0, Nombufs:0
Dropped RX Packets:0
TX Device Packets:0, Bytes:0, Errors:0
Queue Rx:
      Tx:
      Rx Bytes:
      Tx Bytes:
      Errors:
```

The command **`dpdkinfo -x 3`** displays extended statistical information for a specific interface.

```
dpdkinfo -x 3
Driver Name:net_iavf
Interface Info:0000:18:0d.1
Rx Packets:
    rx_good_packets: 6701
    rx_unicast_packets: 0
    rx_multicast_packets: 2987
    rx_broadcast_packets: 3714
    rx_dropped_packets: 0
Tx Packets:
    tx_good_packets: 0
    tx_unicast_packets: 0
    tx_multicast_packets: 0
    tx_broadcast_packets: 0
    tx_dropped_packets: 0
Rx Bytes:
    rx_good_bytes: 1365696
Tx Bytes:
    tx_good_bytes: 0
Errors:
    rx_missed_errors: 0
    rx_errors: 0
    tx_errors: 0
    rx_mbuf_allocation_errors: 0
    inline_ipsec_crypto_ierrors: 0
    inline_ipsec_crypto_ierrors_sad_lookup: 0
    inline_ipsec_crypto_ierrors_not_processed: 0
    inline_ipsec_crypto_ierrors_icv_fail: 0
    inline_ipsec_crypto_ierrors_length: 0
Others:
    inline_ipsec_crypto_ipackets: 0
---------------------------------------------------------------------
```

The `rt` and `nh` Commands

Use the `rt` command to display all routes in a VRF. The `nh` command enables you to inspect the next hops that are known by the vRouter. Next hops tell the vRouter the next location to send a packet in the path to its final destination.

For example, for IPv4 traffic:

```
rt --get 172.68.20.2/32 --vrf 4
Match 172.68.20.2/32 in vRouter inet4 table 0/4/unicast
Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet4 routing table 0/4/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
172.68.20.2/32          0          LPT         16             193        -
```

```
nh --get 193
Id:193        Type:Tunnel         Fmly: AF_INET  Rid:0  Ref_cnt:264        Vrf:0
                Flags:Valid, Policy, MPLSoUDP, Etree Root,
Oif:4 Len:14 Data:88 e6 4b 09 7d 46 40 a6 b7 2c a4 48 08 00 Sip:1.1.1.35 Dip:1.1.24.24
```

For example, for IPv6 traffic:

```
rt --get 2001:172:68:20::/64 --vrf 4 --family inet6
Match 2001:172:68:20::/64 in vRouter inet6 table 0/4/unicast
Flags: L=Label Valid, P=Proxy ARP, T=Trap ARP, F=Flood ARP, Ml=MAC-IP learnt route
vRouter inet6 routing table 0/4/unicast
Destination           PPL        Flags        Label         Nexthop    Stitched MAC(Index)
2001:172:68:20::/64     0          LPT         16             193        -
```

```
nh --get 193
Id:193        Type:Tunnel         Fmly: AF_INET  Rid:0  Ref_cnt:264        Vrf:0
```

```
              Flags:Valid, Policy, MPLSoUDP, Etree Root,
Oif:4 Len:14 Data:88 e6 4b 09 7d 46 40 a6 b7 2c a4 48 08 00 Sip:1.1.1.35 Dip:1.1.24.24
```

The `flow` Command

Use the `flow` command to display all active flows in a system. For example:

```
flow -l --match 169.83.47.170:9398
Flow table(size 161218560, entries 629760)

Entries: Created 162630 Added 162614 Deleted 35136 Changed 35202Processed 162630 Used Overflow
entries 0
(Created Flows/CPU: 0 0 0 0 0 0 0 0 0 0 241 546 15 161828)(oflows 0)

Action:F=Forward, D=Drop N=NAT(S=SNAT, D=DNAT, Ps=SPAT, Pd=DPAT, L=Link Local Port)
 Other:K(nh)=Key_Nexthop, S(nh)=RPF_Nexthop
 Flags:E=Evicted, Ec=Evict Candidate, N=New Flow, M=Modified Dm=Delete Marked
TCP(r=reverse):S=SYN, F=FIN, R=RST, C=HalfClose, E=Established, D=Dead
 Stats:Packets/Bytes

Listing flows matching ([169.83.47.170]:9398)

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------------
   328196<=>524233       169.83.47.170:9398                                  6 (2)
                         172.68.20.20:2159
(Gen: 3, K(nh):206, Action:F, Flags:, TCP:, E:1, QOS:-1, S(nh):206,  Stats:6/360,
 SPort 63929, TTL 0, Sinfo 38.0.0.0)

   524233<=>328196       172.68.20.20:2159                                   6 (2)
                         169.83.47.170:9398
(Gen: 3, K(nh):206, Action:F, Flags:, TCP:, QOS:-1, S(nh):250,  Stats:0/0,
 SPort 60311, TTL 0, Sinfo 0.0.0.0)
```

Troubleshoot via Introspect

IN THIS SECTION

- Introspect | 91

Introspect

For vRouter-agent debugging, we use Introspect. You can access the Introspect data at **http://<host server IP>:8085**. Here is a sample of the Introspect data:
Table 12: Modules shown in contrail-vrouter-agent debug output

Link	and Description
agent.xml	Shows agent operational data. Using this introspect, you can see the list of interfaces, VMs, VNs, VRFs, security groups, ACLs and mirror configurations.
agent_ksync.xml	Shows agent ksync layer for data objects such as interfaces and bridge ports.
agent_profile.xml	shows agent **operdb**, tasks, flows, and statistics summary.
agent_stats_interval.xml	View and set collection period for statistics.
controller.xml	Shows the connection status of the jcnr-controller (cRPD)
cpuinfo.xml	Shows the CPU load and memory usage on the compute node.
ifmap_agent.xml	Shows the current configuration data received from **ifmap**.
kstate.xml	Shows data configured in the vRouter data path.

Table 12: Modules shown in contrail-vrouter-agent debug output *(Continued)*

Link	and Description
mac_learning.xml	Shows entries in vRouter-agent MAC learning table.
sandesh_trace.xml	Gives the different agent module traces such as **oper**, **ksync**, **mac learning**, and **grpc**.
sandesh_uve.xml	Lists all the user visible entitities (UVEs) in the vRouter-agent. The UVEs are used for analytics and telemetry.
stats.xml	Shows vRouter-agent slow path statistics such as error packets, trapped packets, and debug statistics.
task.xml	Shows vRouter-agent worker task details.

NOTE: The table shows grouped output. The cloud-native router does not group or sort the output on live systems.

The **http://*host server IP address*:8085** page displays only a list of HTML links.

6

CHAPTER

Appendix

IN THIS CHAPTER

- Working Samples for Cloud-Native Router Helm Chart | **94**
- Kubernetes Overview | **100**
- Configure Repository Credentials | **102**
- Deploy Prepackaged Images | **103**
- Juniper Technology Previews (Tech Previews) | **104**

Working Samples for Cloud-Native Router Helm Chart

SUMMARY

This topic lists the working samples for the Cloud-Native Router helm chart (values.yaml) for L2, L3, and L2-L3 deployment modes.

IN THIS SECTION

- L2 Only | 94
- L3 Only | 96
- L2-L3 | 97
- Amazon EKS L3 | 98

L2 Only

A working L2 only helm chart sample is shown below:

```
global:
  registry: enterprise-hub.juniper.net/
  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/
      tag: R23.2-154
    crpd:
      repository: junos-docker-local/warthog/
      tag: 23.2R1.14
    jcnrcni:
      repository: junos-docker-local/warthog/
      tag: 23.2-20230628-8cf4350
  fabricInterface:
  - bond0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
  - ens2f2v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
  - ens2f3v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
  - ens1f0v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
  fabricWorkloadInterface:
  - ens1f1v0:
      interface_mode: access
      vlan-id-list: [1110]

  log_level: "INFO"
  log_path: "/var/log/jcnr/"
  syslog_notifications: "/var/log/jcnr/jcnr_notifications.json"

jcnr-vrouter:
  restoreInterfaces: false
  bondInterfaceConfigs:
    - name: "bond0"
      mode: 1             # ACTIVE_BACKUP MODE
      slaveInterfaces:
      - "ens2f0v0"
      - "ens2f1v0"
  mtu: "9000"
  cpu_core_mask: "2,3,22,23"
  stormControlProfiles:
    rate_limit_pf1:
      bandwidth:
        level: 0

  ddp: "auto"
  qosEnable: false
  corePattern: "/var/crash/core-%e.%p.%h.%t"

  coreFilePath: /var/crash
  vrouter_dpdk_uio_driver: "vfio-pci"
```

L3 Only

A working L3 only helm chart sample is shown below:

```
global:
  registry: enterprise-hub.juniper.net/
  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/
      tag: R23.2-156
    crpd:
      repository: junos-docker-local/warthog/
      tag: 23.2R1.14
    jcnrcni:
      repository: junos-docker-local/warthog/
      tag: 23.2-20230628-8cf4350
  fabricInterface:
  - bond34:
      ddp: "auto"
  - ens2f2:
      ddp: "auto"
  - ens1f1:
      ddp: "auto"

  log_level: "INFO"
  log_path: "/var/log/jcnr/"
  syslog_notifications: "/var/log/jcnr/jcnr_notifications.json"

jcnr-vrouter:
  restoreInterfaces: false
  mtu: "9000"
  cpu_core_mask: "2,3,22,23"

  stormControlProfiles:
    rate_limit_pf1:
      bandwidth:
        level: 0
  ddp: "auto"
  qosEnable: false
  corePattern: "/var/crash/core-%e.%p.%h.%t"
  coreFilePath: /var/crash
  vrouter_dpdk_uio_driver: "vfio-pci"
```

L2-L3

A working L2-L3 helm chart sample is shown below:

```
global:
  registry: enterprise-hub.juniper.net/
  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/
      tag: R23.2-154
    crpd:
      repository: junos-docker-local/warthog/
      tag: 23.2R1.14
    jcnrcni:
      repository: junos-docker-local/warthog/
      tag: 23.2-20230628-8cf4350

  fabricInterface:
  - bond0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
      storm-control-profile: rate_limit_pf1
      ddp: "auto"
  - ens2f0v1:
      ddp: "auto"
  - enp179s0f1v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
      ddp: "auto"
  - enp179s0f1v1:
      ddp: "auto"
  log_level: "INFO"
  log_path: "/var/log/jcnr/"
  syslog_notifications: "/var/log/jcnr/jcnr_notifications.json"

jcnr-vrouter:
  restoreInterfaces: false
  bondInterfaceConfigs:
    - name: "bond0"
      mode: 1             # ACTIVE_BACKUP MODE
      slaveInterfaces:
      - "ens2f0v0"
      - "ens2f1v0"
  mtu: "9000"
  cpu_core_mask: "2,3,22,23"

  stormControlProfiles:
    rate_limit_pf1:
      bandwidth:
        level: 0
  ddp: "auto"
  qosEnable: false

  corePattern: "/var/crash/core-%e.%p.%h.%t"
  coreFilePath: /var/crash
  vrouter_dpdk_uio_driver: "vfio-pci"
```

Amazon EKS L3

A working Amazon EKS L3 helm chart sample is shown below:

```
global:
  registry: enterprise-hub.juniper.net/
  repository: jcnr/
  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/
      tag: R23.2-139
    crpd:
      repository: junos-docker-local/warthog/
      tag: 23.2R1.14
    jcnrcni:
      repository: junos-docker-local/warthog/
      tag: master-20230625-7c292ad
  replicas: "7"
  storageClass: gp2
  fabricInterface:
    - subnet: 10.0.3.0/24
      gateway: 10.0.3.1
    - subnet: 10.0.5.0/24
      gateway: 10.0.5.1
    - subnet: 10.0.7.0/24
      gateway: 10.0.7.1
    - subnet: 10.0.8.0/24
      gateway: 10.0.8.1
    - subnet: 10.0.4.0/24
      gateway: 10.0.4.1
    - subnet: 10.0.10.0/24
      gateway: 10.0.10.1
    - subnet: 10.0.6.0/24
      gateway: 10.0.6.1
  log_level: "INFO"
  log_path: "/var/log/jcnr/"
  syslog_notifications: "/var/log/jcnr/jcnr_notifications.json"
  nodeAffinity:
  - key: key1
    operator: In
    values:
    - jcnr
jcnr-vrouter:
  restoreInterfaces: false
  mtu: "9000"
  cpu_core_mask: "2,3"
  stormControlProfiles:
    rate_limit_pf1:
      bandwidth:
        level: 0
  ddp: "auto"
  qosEnable: false
  corePattern: ""
  coreFilePath: /var/crash
  vrouter_dpdk_uio_driver: "vfio-pci"
```

NOTE: For Amazon EKS, you need to additionally update the `dpdkCommandAdditionalArgs` key in the **helmchart/charts/jcnr-vrouter/values.yaml** file and set `tx` and `rx` descriptors to 256. For example:

```
dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 256 --dpdk_rxd_sz 256"
```

Kubernetes Overview

IN THIS SECTION

- Kubernetes Overview | **100**

Kubernetes Overview

NOTE: Juniper Networks refers to primary nodes and backup nodes. Kubernetes refers to master nodes and worker nodes. References in this guide to primary and backup correlate with master and worker in the Kubernetes world.

Kubernetes is an orchestration platform for running containerized applications in a clustered computing environment. It provides automatic deployment, scaling, networking, and management of containerized applications.

A Kubernetes pod consists of one or more containers, with each pod representing an instance of the application. A pod is the smallest unit that Kubernetes can manage. All containers in the pod share the same network name space.

We rely on Kubernetess to orchestrate the infrastructure that the cloud-native router needs to operate. However, we do not supply Kubernetes installation or management instructions in this documentation. See https://kubernetes.io for Kubernetes documentation. Currently, Juniper Cloud-Native Router requires that the Kubernetes cluster be a standalone cluster, meaning that the Kubernetes primary and backup functions both run on a single node.

The major components of a Kubernetes cluster are:

- Nodes

  Kubernetes uses two types of nodes: a primary (control) node and a compute (worker) node. A Kubernetes cluster usually consists of one or more master nodes (in active/standby mode) and one or more worker nodes. You create a node on a physical computer or a virtual machine (VM).

- Pods

  Pods live in nodes and provide a space for containerized applications to run. A Kubernetes pod consists of one or more containers, with each pod representing an instance of the application(s). A pod is the smallest unit that Kubernetes can manage. All containers in a pod share the same network namespace.

- Namespaces

  In Kubernetes, pods operate within a namespace to isolate groups of resources within a cluster. All Kubernetes clusters have a *kube-system* namespace, which is for objects created by the Kubernetes system. Kubernetes also has a *default* namespace, which holds all objects that don't provide their own namespace. The last two preconfigured Kubernetes namespaces are *kube-public* and *kube-node-lease*. The **kube-public** namespace is used to allow authenticated and unauthenticated users to read some aspects of the cluster. Node leases allow the **kubelet** to send heartbeats so that the control plane can detect node failure.

- Kubelet

  The kubelet is the primary node agent that runs on each node. In the case of Juniper Cloud-Native Router, only a single kubelet runs on the cluster since we do not support multinode deployments.

- Containers

  A container is a single package that consists of an entire runtime environment including the application and its:

  - Configuration files
  - Dependencies
  - Libraries
  - Other binaries

  Software that runs in containers can, for the most part, ignore the differences in the those binaries, libraries, and configurations that may exist between the container environment and the environment that hosts the container. Common container types are docker, containerd, and Container Runtime Interface using Open Container Initiative compatible runtimes (CRI-O).

Configure Repository Credentials

SUMMARY

Read this topic to understand how to configure the enterprise-hub.juniper.net repository credentials for Cloud-Native Router installation.

Use this procedure to configure your repository login credentials in your manifests.

1. Install docker if you don't already have docker installed.
2. Log in to the Juniper Networks repository where you pull the container images.

   ```
   docker login enterprise-hub.juniper.net
   ```

   Enter your login credentials when prompted.

   Once you've logged in, your credentials are automatically stored in **~/.docker/config.json**. (If you installed docker using snap, then the credentials are stored in the **~/snap/docker** directory hierarchy.)
3. Encode your credentials in base64 and store the resulting string.

   ```
   ENCODED_CREDS=$(base64 -w 0 config.json)
   ```

   Take a look at the encoded credentials.

   ```
   echo $ENCODED_CREDS
   ```
4. Navigate to `Juniper_Cloud_Native_Router_<release-number>/helmchart` directory. Replace the credentials placeholder in the manifest with the encoded credentials.

   The manifests have a `<base64-encoded-credential>` credentials placeholder. Simply replace the placeholder with the encoded credentials in all manifests.

   ```
   sed -i s/'<base64-encoded-credential>'/$ENCODED_CREDS/ values.yaml
   ```

Double check by searching for the encoded credentials in the manifests.

```
grep $ENCODED_CREDS values.yaml
```

You should see the encoded credentials in the manifests.

Deploy Prepackaged Images

Use this procedure to import Cloud-Native Router images to the container runtime from the downloaded Cloud-Native Router software package .
Your cluster can pull Cloud-Native Router images from the `enterprise-hub.juniper.net` repository or your cluster can use the Cloud-Native Router images that are included in the downloaded Cloud-Native Router software package.

This latter option is useful if your cluster doesn't have access to the Internet or if you want to set up your own repository.

Setting up your own repository is beyond the scope of this document, but your cluster can still use the included images if you manually import them to the container runtime on each cluster node running JCNR. Simply use the respective container runtime commands. We show you how to do this in the procedure below.

1. Locate the images tarball in the **Juniper_Cloud_Native_Router_*<release>*/images** directory.

   The images tarball is in a gzipped file (**jcnr-images.tar.gz**).
2. Copy the gzipped images tarball to every node where you're installing JCNR.
3. SSH to one of the nodes and go to the directory where you copied the gzipped images tarball.
4. Gunzip the gzipped images tarball that you just copied over.

   ```
   gunzip jcnr-images.tar.gz
   ```

   ```
   ls
   jcnr-images.tar
   ```
5. Import the images to the container runtime.
   - containerd: `ctr -n k8s.io images import jcnr-images.tar`
   - docker: `docker load -i jcnr-images.tar`
6. Check that the images have been imported.

- containerd: `ctr -n k8s.io images ls`
- docker: `docker images`

7. Repeat steps 3 to 6 on each node where you're installing JCNR.

When you install Cloud-Native Router later on, the cluster first searches locally for the required images before reaching out to `enterprise-hub.juniper.net`. Since you manually imported the images locally on each node, the cluster finds the images locally and does not need to download them from an external source.

Juniper Technology Previews (Tech Previews)

Tech Previews enable you to test functionality and provide feedback during the development process of innovations that are not final production features. The goal of a Tech Preview is for the feature to gain wider exposure and potential full support in a future release. Customers are encouraged to provide feedback and functionality suggestions for a Technology Preview feature before it becomes fully supported.

Tech Previews may not be functionally complete, may have functional alterations in future releases, or may get dropped under changing markets or unexpected conditions, at Juniper's sole discretion. Juniper recommends that you use Tech Preview features in non-production environments only.

Juniper considers feedback to add and improve future iterations of the general availability of the innovations. Your feedback does not assert any intellectual property claim, and Juniper may implement your feedback without violating your or any other party's rights.

These features are "as is" and voluntary use. Juniper Support will attempt to resolve any issues that customers experience when using these features and create bug reports on behalf of support cases. However, Juniper may not provide comprehensive support services to Tech Preview features. Certain features may have reduced or modified security, accessibility, availability, and reliability standards relative to General Availability software. Tech Preview features are not eligible for P1/P2 JTAC cases, and should not be subject to existing SLAs or service agreements.

For additional details, please contact Juniper Support or your local account team.