Juniper Networks | Engineering Simplicity

Juniper Cloud-Native Router Deployment Guide

Published
2025-04-08

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

YEAR 2000 NOTICE

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

END USER LICENSE AGREEMENT

Table of Contents

1

CHAPTER

Introduction

IN THIS CHAPTER

Juniper Cloud-Native Router Overview

SUMMARY

This topic provides an overview of the Juniper Cloud-Native Router (JCNR) overview, use cases, and features.

IN THIS SECTION

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 Data Plane Development Kit (DPDK) or extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath based vRouter forwarding plane. It is implemented in Kubernetes and interacts seemlessly with a Kubernetes container network interface (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) or eBPF XDP datapath enabled vRouter as a forwarding plane, providing packet forwarding for 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 router 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.

The extended Berkley Packet Filter (eBPF) is a Linux kernel technology that executes user-defined programs inside a sandbox virtual machine. It enables low-level networking programs to execute with optimal performance. The eXpress Data Path (XDP) frameworks enables high-speed packet processing for the eBPF programs. Cloud-Native Router supports eBPF XDP datapath based vRouter.

In this integrated solution, the Cloud-Native Router 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.

- Run routing protocols such as ISIS, BGP, and OSPF and much more.

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.
- Out-of-the-box software-based open radio access network (O-RAN) support.
- Quick spin up with containerized deployment.
- Highly scalable solution.
- L3 features such as transit gateway, support for routing protocols, BFD, VRRP, VRF-Lite, EVPN Type-5, ECMP and BGP Unnumbered, access control lists, SRv6.
- L2 functionality, such as MAC learning, MAC aging, MAC limiting, native VLAN, L2 statistics, and access control lists (ACLs).
- 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 and bridge domains.
- Trunk and access ports.
- Support for multiple virtual functions (VF) on Ethernet NICs.
- Support for bonded VF interfaces.
- Rate limiting of egress broadcast, unknown unicast, and multicast traffic on fabric interfaces.
- IPv4 and IPv6 routing.

Juniper Cloud-Native Router Components

SUMMARY

The Juniper Cloud-Native Router solution consists of several components including the Cloud-Native Router controller, the Data Plane Development Kit (DPDK) or extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath based Cloud-Native Router vRouter and the JCNR-CNI. This topic provides a brief overview of the components of the Juniper Cloud-Native Router.

IN THIS SECTION

Cloud-Native Router Components

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

Figure 1 on page 6 shows the components of the Juniper Cloud-Native Router inside a Kubernetes cluster when implemented with DPDK based vRouter.

Figure 1: Components of Juniper Cloud-Native Router (DPDK Datapath)

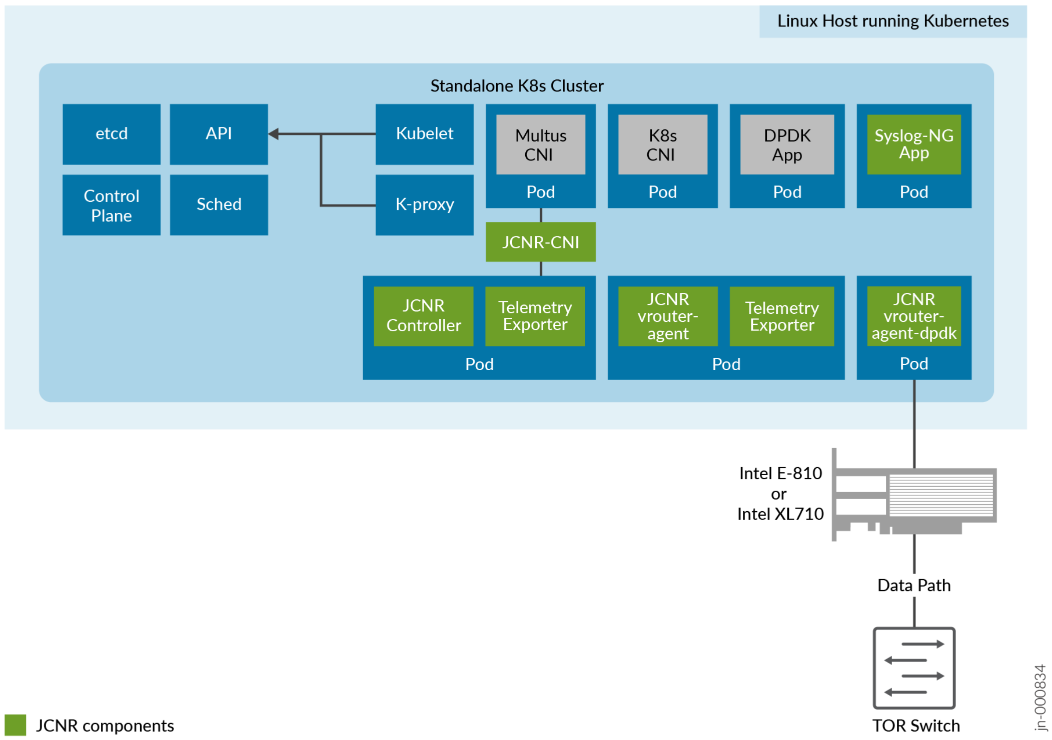
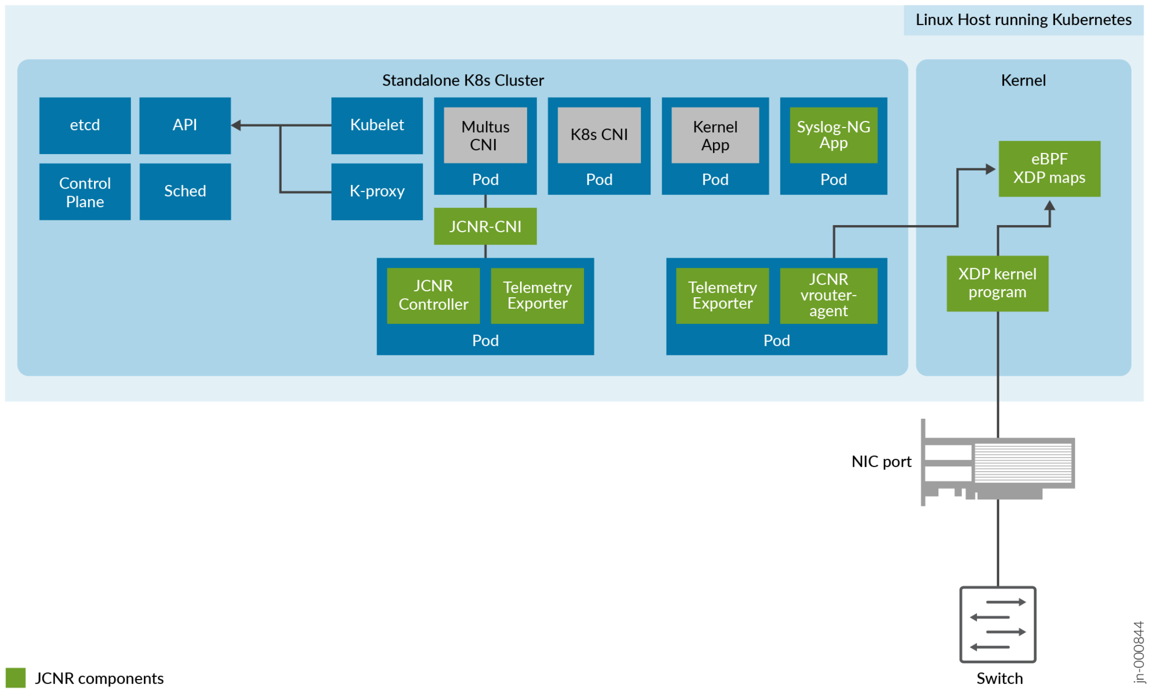


Figure 2 on page 7 shows the components of the Juniper Cloud-Native Router inside a Kubernetes cluster when implemented with eBPF XDP based vRouter.

Figure 2: Components of Juniper Cloud-Native Router (eBPF XDP Datapath)



Cloud-Native Router Controller

The Cloud-Native Router Controller is the control-plane of the cloud-native router solution that runs the Junos containerized routing protocol Daemon (cRPD). It is implemented 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 the Cloud-Native Router vRouter and other components for implementation.

For example, firewall filters (ACLs) configured on the controller are sent to the Cloud-Native Router 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, IS-IS and OSPF.
- 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, establishing peer relationships with other BGP speakers to exchange routing information.
- Exports control plane telemetry data to Prometheus and gNMI.

Configuration Options

Use the "configlet resource" on page 62 to configure the cRPD pods.

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. The vRouter functionality is implemented in two pods, one for the vrouter-agent and the vrouter-telemetry-exporter, and the other for the vrouter-agent-dpdk. This split gives you the flexibility to tailor CPU resources to the different vRouter components as needed.

The vRouter supports both Data Plane Development Kit (DPDK) and extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath.

NOTE: Cloud-Native Router eBPF XDP Datapath is a *Juniper Technology Preview (Tech Preview)* feature. Limited features are supported. See "Juniper Cloud-Native Router vRouter Datapath" on page 11 for more details.

Cloud-Native Router vRouter Functionality:

- Performs routing with Layer 3 virtual private networks.

- Performs L2 forwarding.
- Supports high-performance DPDK-based forwarding.
- Supports high performance eBPF XDP datapath based forwarding.
- Exports data plane telemetry data to Prometheus and gNMI.

Benefits of vRouter:

- High-performance packet processing.
- 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 controller 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.
  - creating L3 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 Kernel networking stack.
- Creating pod interfaces in access or trunk mode.
- Attaching pod interfaces to bridge domains and virtual routers.
- Supporting IPAM plug-in for Dynamic IP address allocation.
- Allocating unique socket interfaces for virtio interfaces.
- Managing the networking tasks in pods such as assigning IP addresses and setting up of interfaces between the pod and vRouter in a 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 the controller 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 the controller.

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.

Juniper Cloud-Native Router vRouter Datapath

SUMMARY

Cloud-Native Router supports both Data Plane Development Kit (DPDK) and extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath based vRouter forwarding plane.

IN THIS SECTION

The Cloud-Native Router vRouter forwarding plane supports both the Data Plane Development Kit (DPDK) and extended Berkley Packet Filter (eBPF) eXpress Data Path (XDP) datapath for high-speed packet processing.

Data Plane Development Kit (DPDK)

DPDK is an open-source set of libraries and drivers for rapid packet processing. 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. This method of packet routing lets the application poll for packets, which prevents the overhead of interrupts from the NIC.

DPDK's poll mode drivers (PMDs) use the physical interface (NIC) of a VM's host instead of the Linux kernel's interrupt-based drivers. The NIC's registers operate in user space, which makes them accessible by DPDK’s PMDs. As a result, the host OS does not need to manage the NIC's registers. This means that the DPDK application manages all packet polling, packet processing, and packet forwarding of a NIC. Instead of waiting for an I/O interrupt to occur, a DPDK application constantly polls for packets and processes these packets immediately upon receiving them.

DPDK datapath has high CPU usage due to the poll mode and has high maintenance costs. Also, when implementing DPDK, the NIC is no longer available in the kernel, hence sockets and forwarding plane code must be re-implemented.

eBPF XDP

NOTE: This is a *Juniper Technology Preview (Tech Preview)* feature.

Cloud-Native Router also supports an eBPF XDP datapath based vRouter. eBPF (extended Berkley Packet Filter) is a Linux kernel technology that executes user-defined programs inside a sandbox virtual machine. It enables low-level networking programs to execute with optimal performance. The eXpress Data Path (XDP) frameworks enables high-speed packet processing for the eBPF programs. Cloud-Native Router supports XDP in native (driver) mode on Baremental server deployments for limited drivers only. Please see the "System Requirements" on page 36 for more details.

Benefits of eBPF XDP Datapath

Benefits of eBPF XDP Datapath include:

- An eBPF XDP kernel program and its custom library is easier to maintain across kernel versions and has wider kernel compatibility. The kernel dependencies are limited to a small set of eBPF helper functions.
- The program is safer since it is analysed by the in-built Linux eBPF verifier before it is loaded into the kernel.
- Offers higher performance using kernel bypass and omitting socket buffer (skb) allocation.

Supported Cloud-Native Router Features for eBPF XDP

The following Cloud-Native Router Features are supported with eBPF XDP for IPv4 traffic only:

- L3 traffic with Cloud-Native Router deployed as a sending, receiving or transit router
- VRF-Lite
- MPLSoUDP
- IGPs—OSPF, IS-IS
- BGP route advertisements

NOTE: When deploying JCNR, you can configure the `agentModeType` attribute in the helmchart to select either a DPDK based or eBPF XDP datapath based vRouter.

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

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.

> **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.

Cloud-Native Router Interfaces Overview

SUMMARY

This topic provides information on the network communication interfaces provided by the JCNR-Controller. Fabric interfaces are aggregated interfaces that receive traffic from multiple interfaces. Interfaces to which different workloads are connected are called workload interfaces.

IN THIS SECTION

Read this topic to understand the network communication interfaces provided by the JCNR-Controller. We cover interface names, what they connect to, how they communicate and the services they provide.

Juniper Cloud-Native Router Interface Types

Juniper Cloud-Native Router supports two types of interfaces:

- **Fabric interfaces**—Aggregated interfaces that receive traffic from multiple interfaces. Fabric interfaces are always physical interfaces. They can either be a physical function (PF) or a virtual function (VF). The throughput requirement for these interfaces is higher, hence multiple hardware queues are allocated to them. Each hardware queue is allocated with a dedicated CPU core . The interfaces are configured for the cloud-native router using the appropriate `values.yaml` file in the deployer helmcharts. You can view the interface mapping using the `dpdkinfo -c` command (View the *Troubleshoot using the vRouter CLI* topic for more details). You also have fabric workload interfaces that have low throughput requirement. Only one hardware queue is allocated to the interface, thereby saving precious CPU resources. These interfaces can be configured using the appropriate `values.yaml` file in the deployer helmcharts.
- **Workload interfaces**—Interfaces to which different workloads are connected. They can either be software-based or hardware-based interfaces. Software-based interfaces (pod interfaces) are either high-performance interfaces using the Data Plane Development Kit (DPDK) poll mode driver (PMD) or a low-performance interfaces using the kernel driver. Typically the DPDK interfaces are used for data traffic such as the GPRS Tunneling Protocol for user data (GTP-U) traffic and the kernel-based

interfaces are used for control plane data traffic such as TCP. The kernel pod interfaces are typically for the operations, administration and maintenance (OAM) traffic or are used by non-DPDK pods. The kernel pod interfaces are configured as a veth-pair, with one end of the interface in the pod and the other end in the Linux kernel on the host. The DPDK native pod interfaces (virtio interfaces) are plumbed as vhost-user interfaces to the DPDK vRouter by the CNI. Cloud-Native Router also supports bonded interfaces via the link bonding PMD. These interfaces can be configured using the appropriate `values.yaml` file in the deployer helmcharts.

Cloud-Native Router supports different types of VLAN interfaces including trunk, access and sub-interfaces across fabric and workload interfaces.

Cloud-Native Router Interface Details

The different Cloud-Native Router interfaces are provided in detail below:

Agent Interface

The vRouter has only one agent interface. The agent interface enables communication between the vRouter-agent and the vRouter containers. On the vRouter CLI when you issue the `vif --list` command, the agent interface looks like this:

```
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:650  bytes:99307 errors:0
            Drops:0
```

L3 Fabric Interface (DPDK)

A layer-3 fabric interface bound to the DPDK.

L3 fabric interface in cRPD can be reviewed on the cRPD shell using the junos `show interfaces` command:

```
show interfaces routing ens2f2
Interface        State Addresses
ens2f2           Up    MPLS  enabled
                       ISO   enabled
```

```
                        INET  192.21.2.4
                        INET6 2001:192:21:2::4
                        INET6 fe80::c5da:7e9c:e168:56d7
                        INET6 fe80::a0be:69ff:fe59:8b58
```

The corresponding physical and tap interfaces can be seen on the vRouter using the `vif --list` command on the vRouter shell.

```
vif0/1      PCI: 0000:17:01.1 (Speed 25000, Duplex 1) NH: 7 MTU: 9000 <- PCI
Address
            Type:Physical HWaddr:d6:93:87:91:45:6c IPaddr: 192.21.2.4 <- Physical interface
            IP6addr:2001:192:21:2::4 <- IPv6 address
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:2 Flags:L3L2Vof QOS:0 Ref:16 <- L3 (only) interface
            RX port   packets:423168341 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:17:01.1  Status: UP  Driver: net_iavf
            RX packets:423168341  bytes:29123418594 errors:0
            TX packets:417508247  bytes:417226216530 errors:0
            Drops:8
            TX port   packets:417508247 errors:0
```

```
vif0/2      PMD: ens2f2 NH: 12 MTU: 9000 <- Tap interface name as seen by cRPD
            Type:Host HWaddr:d6:93:87:91:45:6c IPaddr: 192.21.2.4 <- Tap interface type
            IP6addr:2001:192:21:2::4
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:15 TxXVif:1  <-cross-connected to
vif 1
            RX device packets:306995  bytes:25719830 errors:0
            RX queue  packets:306995 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:306995  bytes:25719830 errors:0
            TX packets:307489  bytes:25880250 errors:0
            Drops:0
            TX queue  packets:307489 errors:0
            TX device packets:307489  bytes:25880250 errors:0
```

L3 Bond Interface (DPDK)

A layer 3 bond interface bound to DPDK.

```
show interfaces routing bond34
Interface        State Addresses
bond34           Up    INET6 2001:192:7:7::4
                       ISO   enabled
                       INET  192.7.7.4
                       INET6 fe80::527c:6fff:fe48:7574
```

```
vif0/3      PCI: 0000:00:00.0 (Speed 25000, Duplex 1) NH: 6 MTU: 1514 <- Bond interface (PCI id 
0)
            Type:Physical HWaddr:50:7c:6f:48:75:74 IPaddr:192.7.7.4 <- Physical interface
            IP6addr:2001:192:7:7::4
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:1 Flags:TcL3L2Vof QOS:0 Ref:18
            RX port   packets:402183888 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: eth_bond_bond34  Status: UP  Driver: net_bonding <- Bonded master
            Slave Interface(0): 0000:5e:00.0  Status: UP  Driver: net_ice <- Bond slave - 1
            Slave Interface(1): 0000:af:00.0  Status: UP  Driver: net_ice <- Bond slave - 2
            RX packets:402183888  bytes:49519387070 errors:0
            TX packets:79226  bytes:7330912 errors:0
            Drops:1393
            TX port   packets:79226 errors:0
```

```
vif0/4      PMD: bond34 NH: 11 MTU: 9000
            Type:Host HWaddr:50:7c:6f:48:75:74 IPaddr:192.7.7.4 <- Tap interface
            IP6addr:2001:192:7:7::4
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:15 TxXVif:3 <- Tap interface for 
bond
            RX device packets:76357  bytes:7101918 errors:0
            RX queue  packets:76357 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:76357  bytes:7101918 errors:0
            TX packets:75349  bytes:6946908 errors:0
            Drops:0
```

```
            TX queue  packets:75349 errors:0
            TX device packets:75349  bytes:6946908 errors:0
```

L3 Pod VLAN Sub-Interface (DPDK)

Starting in Juniper Cloud-Native Router Release 23.2, the cloud-native router supports the use of VLAN sub-interfaces in L3 mode, bound to DPDK.

Corresponding interface state in cRPD:

```
show interfaces routing ens1f0v1.201
Interface          State Addresses
ens1f0v1.201       Up    MPLS  enabled
                   ISO   enabled
                   INET6 fe80::b89c:fff:feab:e2c9
```

```
vif0/2      PCI: 0000:17:01.1 (Speed 25000, Duplex 1) NH: 7 MTU: 9000
            Type:Physical HWaddr:d6:93:87:91:45:6c IPaddr:0.0.0.0
            IP6addr:fe80::d493:87ff:fe91:456c <- IPv6 address
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:2 Flags:L3L2Vof QOS:0 Ref:16 <- L3 (only) interface
            RX port   packets:423168341 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:17:01.1  Status: UP  Driver: net_iavf
            RX packets:423168341  bytes:29123418594 errors:0
            TX packets:417508247  bytes:417226216530 errors:0
            Drops:8
            TX port   packets:417508247 errors:0
```

```
vif0/5      PMD: ens1f0v1 NH: 12 MTU: 9000
            Type:Host HWaddr:d6:93:87:91:45:6c IPaddr:0.0.0.0
            IP6addr:fe80::d493:87ff:fe91:456c
            DDP: OFF SwLB: ON
            Vrf:2 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:15 TxXVif:2 <- L3 (only) tap
interface
            RX device packets:306995  bytes:25719830 errors:0
            RX queue  packets:306995 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:306995  bytes:25719830 errors:0
```

```
            TX packets:307489  bytes:25880250
errors:0

            Drops:0
            TX queue  packets:307489 errors:0
            TX device packets:307489  bytes:25880250 errors:0
```

```
vif0/9      Virtual: ens1f0v1.201 Vlan(o/i)(,S): 201/201 Parent:vif0/2 NH: 36 MTU: 1514 <- VLAN
fabric sub-intf with parent as vif 2 and VLAN tag as 201
            Type:Virtual(Vlan) HWaddr:d6:93:87:91:45:6c IPaddr:103.1.1.2
            IP6addr:fe80::d493:87ff:fe91:456c
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:1 Flags:L3DProxyEr QOS:-1 Ref:4
            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
```

```
vif0/10     Virtual: ens1f0v1.201 Vlan(o/i)(,S): 201/201 Parent:vif0/5 NH: 21 MTU: 9000
            Type:Virtual(Vlan) HWaddr:d6:93:87:91:45:6c IPaddr:103.1.1.2
            IP6addr:fe80::d493:87ff:fe91:456c
            DDP: OFF SwLB: ON
            Vrf:1 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:4 TxXVif:9 <- VLAN tap sub-intf
cross connected to fabric sub-intf vif 9 and parent as tap intf vif 5
            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
```

```
vif0/50     PMD: vhostnet1-9403fd77-648a-47 NH: 177 MTU: 9160                     ---> pod
interface
            Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:65535 Mcast Vrf:65535 Flags:L3DProxyEr QOS:-1 Ref:20
            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
```

```
vif0/51   Virtual: vhostnet1-9403fd77-648a-47.201 Vlan(o/i)(,S): 201/201 NH: 17 MTU: 1514
          Parent:vif0/50                                                       ---->L3 pod
sub-interface, parent is the pod interface
          Type:Virtual(Vlan) HWaddr:00:00:5e:00:01:00 IPaddr:99.62.0.2
          IP6addr:1234::633e:2
          DDP: OFF SwLB: ON
          Vrf:2 Mcast Vrf:2 Flags:PL3DProxyEr QOS:-1 Ref:4
          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
```

L3 Pod Kernel Interface

These are non-DPDK L3 pod interfaces. Interface state in the cRPD:

```
show interfaces routing jvknet1-0af476e
Interface        State Addresses
jvknet1-0af476e  Up    INET6 enabled
                       INET6 abcd:2:51:1::4
                       ISO   enabled
                       INET  enabled
                       INET  2.51.1.4
```

```
vif0/13   Ethernet: jvknet1-0af476e NH: 35 MTU: 9160 <- Kernel interface (jvk) of CNF
          Type:Virtual HWaddr:00:00:5e:00:01:00 IPaddr:2.51.1.4 <- pod/ workload
          IP6addr:abcd:2:51:1::4
          DDP: OFF SwLB: ON
          Vrf:1 Mcast Vrf:1 Flags:PL3DVofProxyEr QOS:-1 Ref:11
          RX port   packets:47 errors:0
          RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
          RX packets:47  bytes:13012 errors:0
          TX packets:0  bytes:0 errors:0
          Drops:47
```

L2 Fabric Interface (DPDK, Physical Trunk)

DPDK L2 fabric interfaces, which are associated with the physical network interface card (NIC) on the host server, accept traffic from multiple VLANs. The trunk interfaces accept only tagged packets. Any untagged packets are dropped. These interfaces can accept a VLAN filter to allow only specific VLAN packets. A trunk interface can be a part of multiple bridge-domains (BD). A bridge domain is a set of logical ports that share the same flooding or broadcast characteristics. Like a VLAN, a bridge domain spans one or more ports of multiple devices.

The cRPD interface configuration using the `show configuration` command looks like this (the output is trimmed for brevity):

```
interfaces {
    ens786f0v0 {
        unit 0 {
            family bridge {
                interface-mode trunk;
                vlan-id-list 1001-1100;
            }
        }
    }
}
```

On the vRouter CLI when you issue the `vif --list` command, the DPDK VF fabric interface looks like this:

```
vif0/1    PCI: 0000:31:01.0 (Speed 10000, Duplex 1)
          Type:Physical HWaddr:d6:22:c5:42:de:c3
          Vrf:65535 Flags:L2Vof QOS:-1 Ref:12
          RX queue packets:11813 errors:1
          RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 1 0
          Fabric Interface: 0000:31:01.0 Status: UP Driver: net_iavf
          Vlan Mode: Trunk Vlan: 1001-1100
          RX packets:0 bytes:0 errors:49962
          TX packets:18188356 bytes:2037400554 errors:0
          Drops:49963
```

DPDK L2 Bond Interface (Active-Standby, Trunk)

Layer-2 Bond interfaces accept traffic from multiple VLANs. A bond interface runs in the active or standby mode (mode 0). You define the bond interface in the helm chart configuration as follows:

```
bondInterfaceConfigs:
- name: "bond0"
  mode: 1          # ACTIVE_BACKUP MODE
  slaveInterfaces:
  - "ens2f0v1"
  - "ens2f1v1"
```

```
 - bond0:
      ddp: "auto"
      interface_mode: trunk
      vlan-id-list: [1001-1100]
      storm-control-profile: rate_limit_pf1
      native-vlan-id: 1001
      no-local-switching: true
```

The cRPD interface configuration using the `show configuration` command looks like this (the output is trimmed for brevity):

```
interfaces {
    bond0 {
         unit 0 {
             family bridge
             interface-mode trunk;
             vlan-id-list 1001-1100;
         }
    }
}
```

On the vRouter CLI when you issue the `vif --list` command, the bond interface looks like this:

```
vif0/2      PCI: 0000:00:00.0 (Speed 10000, Duplex 1)
            Type:Physical HWaddr:32:f8:ad:8c:d3:bc
            Vrf:65535 Flags:L2Vof QOS:-1 Ref:8
            RX queue  packets:1882 errors:0
```

```
              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): 0000:81:01.0  Status: UP  Driver: net_iavf
              Slave Interface(1): 0000:81:03.0  Status: UP  Driver: net_iavf
              Vlan Mode: Trunk  Vlan: 1001-1100
              RX packets:8108366000  bytes:486501960000 errors:4234
              TX packets:65083776  bytes:4949969408 errors:0
              Drops:8108370394
```

DPDK L2 Pod Interface (Virtio Trunk)

The trunk interfaces accept only tagged packets. Any untagged packets are dropped. These interfaces can accept a VLAN filter to allow only specific VLAN packets. A trunk interface can be a part of multiple bridge-domains (BD). A bridge domain is a set of logical ports that share the same flooding or broadcast characteristics. Like a VLAN, a bridge domain spans one or more ports of multiple devices. Virtio interfaces are associated with pod interfaces that use virtio on the DPDK data plane.

The cRPD interface configuration using the `show configuration` command looks like this (the output is trimmed for brevity):

```
interfaces {
    vhost242ip-93883f16-9ebb-4acf-b {
         unit 0 {
             family bridge {
                 interface-mode trunk;
                 vlan-id-list 1001-1003;
             }
         }
    }
}
```

On the vRouter CLI when you issue the `vif --list` command, the virtio with DPDK data plane interface looks like this:

```
vif0/3     PMD: vhost242ip-93883f16-9ebb-4acf-b
           Type:Virtual HWaddr:00:16:3e:7e:84:a3
           Vrf:65535 Flags:L2 QOS:-1 Ref:13
           RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
           Vlan Mode: Trunk Vlan: 1001-1003
           RX packets:0 bytes:0 errors:0
           TX packets:10604432 bytes:1314930908 errors:0
```

```
          Drops:0
          TX port packets:0 errors:10604432
```

L2 Pod Kernel Interface (Access)

The access interfaces accept both tagged and untagged packets. Untagged packets are tagged with the access VLAN or access BD. Any tagged packets other than the ones with access VLAN are dropped. The access interfaces is a part of a single bridge-domain. It does not have any parent interface.

The cRPD interface configuration using the `show configuration` command looks like this (the output is trimmed for brevity):

```
routing-instances {
    switch {
         instance-type virtual-switch;
         bridge-domains
{

             bd1001 {
                 vlan-id 1001;
                 interface jvknet1-eed79ff;
             }
         }
    }
}
```

On the vRouter CLI when you issue the `vif --list` command, the veth pair interface looks like this:

```
vif0/4      Ethernet: jvknet1-88c44c3
            Type:Virtual HWaddr:02:00:00:3a:8f:73
            Vrf:0 Flags:L2Vof QOS:-1 Ref:10
            RX queue packets:524 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Vlan Mode: Access Vlan Id: 1001 OVlan Id: 1001
            RX packets:9 bytes:802 errors:515
            TX packets:0 bytes:0 errors:0
            Drops: 525
```

L2 Pod VLAN Sub-interface (DPDK)

You can configure a user pod with a Layer 2 VLAN sub-interface and attach it to the Cloud-Native Router instance. VLAN sub-interfaces are like logical interfaces on a physical switch or router. They access only tagged packets that match the configured VLAN tag. A sub-interface has a parent interface. A parent interface can have multiple sub-interfaces, each with a VLAN ID. When you run the cloud-native router, you must associate each sub-interface with a specific VLAN.

The cRPD interface configuration viewed using the `show configuration` command is as shown below (the output is trimmed for brevity).

For **L2**:

```
routing-instances {
    switch {
         instance-type virtual-switch;
         bridge-domains
{

             bd3003 {
                 vlan-id 3003;
                 interface vhostnet1-71cd7db1-1a5e-49.3003;
             }
         }
     }
}
```

On the vRouter, a VLAN sub-interface configuration is as shown below:

```
vif0/4      PMD: vhostnet1-71cd7db1-1a5e-49 MTU: 9160
            Type:Virtual HWaddr:02:00:00:84:dc:42
            DDP: OFF SwLB: ON
            Vrf:65535 Flags:L2 QOS:-1 Ref:14
            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
            TX port   packets:0 errors:293

vif0/5      Virtual: vhostnet1-71cd7db1-1a5e-49.3003 Vlan(o/i)(,S): 3003/3003 Parent:vif0/4
            Type:Virtual(Vlan) HWaddr:00:99:99:99:33:09
            Vrf:0 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:0  bytes:0 errors:0
            Drops:0
```

RELATED DOCUMENTATION

Cloud-Native Router Use-Cases and Configuration Overview

2

CHAPTER

Install Cloud-Native Router on Baremetal Server

IN THIS CHAPTER

Install and Verify Juniper Cloud-Native Router for Baremetal Servers

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

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 "System Requirements for Baremetal Servers" on page 36 section to ensure the cluster has all the required configuration.
2. Download the desired Cloud-Native Router software package to the directory of your choice.

   You have the option of downloading the package to install Cloud-Native Router only or downloading the package to install JNCR together with Juniper cSRX. See "Cloud-Native Router Software Download Packages" on page 387 for a description of the packages available. If you don't want to install Juniper cSRX now, you can always choose to install Juniper cSRX on your working Cloud-Native Router installation later.
3. Expand the downloaded package.

   ```
   tar xzvf <sw_package>.tar.gz
   ```
4. Change directory to the main installation directory.

   - If you're installing Cloud-Native Router only, then:

     ```
     cd Juniper_Cloud_Native_Router_<release>
     ```

This directory contains the Helm chart for Cloud-Native Router only.

- If you're installing Cloud-Native Router and cSRX at the same time, then:

```
cd Juniper_Cloud_Native_Router_CSRX_<release>
```

This directory contains the combination Helm chart for Cloud-Native Router and cSRX.

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

5. View the contents in the current directory.

```
ls
helmchart  images  README.md  secrets
```

6. Change to the **helmchart** directory and expand the Helm chart.

```
cd helmchart
```

- For Cloud-Native Router only:

```
ls
jcnr-<release>.tgz
```

```
tar -xzvf jcnr-<release>.tgz
```

```
ls
jcnr   jcnr-<release>.tgz
```

The Helm chart is located in the **jcnr** directory.

- For the combined Cloud-Native Router and cSRX:

```
ls
jcnr_csrx-<release>.tgz
```

```
tar -xzvf jcnr_csrx-<release>.tgz
```

```
ls
jcnr_csrx   jcnr_csrx-<release>.tgz
```

  The Helm chart is located in the **jcnr_csrx** directory.

7. 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 398 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 399 for instructions on how to import images to the local container runtime.

8. Follow the steps in "Installing Your License" on page 353 to install your Cloud-Native Router license.

9. 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**.

10. Apply **secrets/jcnr-secrets.yaml** to the cluster.

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

11. If desired, configure how cores are assigned to the vRouter DPDK containers. See "Allocate CPUs to the Cloud-Native Router Forwarding Plane" on page 355.
12. Customize the Helm chart for your deployment using the **helmchart/jcnr/values.yaml** or **helmchart/jcnr_csrx/values.yaml** file.

    See "Customize Cloud-Native Router Helm Chart for Bare Metal Servers" on page 48 for descriptions of the Helm chart configurations.
13. Optionally, customize Cloud-Native Router configuration.

    See "Customize Cloud-Native Router Configuration " on page 62 for creating and applying the cRPD customizations.
14. If you're installing Juniper cSRX now, then follow the procedure in "Apply the cSRX License and Configure cSRX" on page 338.
15. Label the nodes where you want Cloud-Native Router to be installed based on the `nodeaffinity` configuration (if defined in the `values.yaml`). For example:

```
kubectl label nodes ip-10.0.100.17.lab.net key1=jcnr --overwrite
```

16. Deploy the Juniper Cloud-Native Router using the Helm chart.

    Navigate to the **helmchart/jcnr** or the **helmchart/jcnr_csrx** directory and run the following command:

```
helm install jcnr .
```

or

```
helm install jcnr-csrx .
```

```
NAME: jcnr
LAST DEPLOYED: Fri Dec 22 06:04:33 2023
NAMESPACE: default
STATUS: deployed
```

```
REVISION: 1
TEST SUITE: None
```

17. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

Sample output:

```
NAME    NAMESPACE    REVISION    UPDATED          STATUS      CHART             APP VERSION
jcnr    default      1           <date-time>      deployed    jcnr-<version>    <version>
```

Verify Installation

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

> **NOTE**: The output shown in this example procedure is affected by the number of nodes in the cluster. The output you see in your setup may differ in that regard.

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         jcnr-0-dp-contrail-vrouter-nodes-b2jxp         2/2     Running   0            87s
contrail         jcnr-0-dp-contrail-vrouter-nodes-vrdpdk-g7wrk  1/1     Running   0            87s
jcnr             jcnr-0-crpd-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   0            4h2m
kube-system      calico-node-28w98                              1/1     Running   0            86d
kube-system      coredns-54bf8d85c7-vkpgs                       1/1     Running   0            3h8m
```

```
kube-system      dns-autoscaler-7944dc7978-ws9fn            1/1     Running   0                 86d
kube-system      kube-apiserver-ix-esx-06                   1/1     Running   0                 86d
kube-system      kube-controller-manager-ix-esx-06          1/1     Running   0                 86d
kube-system      kube-multus-ds-amd64-jl69w                 1/1     Running   0                 86d
kube-system      kube-proxy-qm5bl                           1/1     Running   0                 86d
kube-system      kube-scheduler-ix-esx-06                   1/1     Running   0                 86d
kube-system      nodelocaldns-bntfp                         1/1     Running   0                 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      jcnr-0-dp-contrail-vrouter-nodes         1         1         1       1            1           <none>                     90m
contrail      jcnr-0-dp-contrail-vrouter-nodes-vrdpdk  1         1         1       1            1           <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        jcnr-0-crpd   1/1     27m
```

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

   a. View the *Access cRPD CLI* section for instructions 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 *Access vRouter CLI* section for instruction 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 0 0 0 0 0 0 0 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.

System Requirements for Baremetal Servers

IN THIS SECTION

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

Minimum Host System Requirements for Bare Metal

Table 1 on page 37 and Table 2 on page 39 list the host system requirements for installing Cloud-Native Router on bare metal servers.

Table 1: Minimum Host System Requirements (DPDK) for Bare Metal

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, 8.7, 8.8, 8.9, 9.3
	Ubuntu	22.04.4 LTS
Kernel Version	RedHat Enterprise Linux (RHEL): 4.18.X Rocky Linux: 4.18.X, 5.14.X Ubuntu: 5.15.X	
NIC	• Intel E810 CVL with Firmware 4.22 0x8001a1cf 1.3346.0 • Intel E810 CPK with Firmware 2.20 0x80015dc1 1.3083.0 • Intel E810-CQDA2 with Firmware 4.20 0x80017785 1.3346.0 • Intel XL710 with Firmware 9.20 0x8000e0e9 0.0.0 • Mellanox ConnectX-6 • Mellanox ConnectX-7	Support for Mellanox NICs is considered a Juniper Technology Preview ("Tech Preview" on page 452) feature. When using Mellanox NICs, ensure your interface names do not exceed 11 characters in length.
IAVF driver	4.8.2	

Table 1: Minimum Host System Requirements (DPDK) for Bare Metal ***(Continued)***

Component	Value/Version	Notes
ICE_COMMS	1.3.35.0	
ICE	1.11.20.13	ICE driver is used only with the Intel E810 NIC
i40e	2.22.18.1	i40e driver is used only with the Intel XL710 NIC
Kubernetes (K8s)	1.22.x, 1.23.x, 1.25x, 1.28.x, 1.29.x	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).
Calico	3.22.x	
Multus	3.8	
Helm	3.9.x	
Container-RT	containerd 1.6.x, 1.7.x	Other container runtimes may work but have not been tested with JCNR.
NOTE: The component versions listed in this table are expected to work with JCNR, but not every version or combination is tested in every release.		

Table 2: Minimum Host System Requirements (eBPF) for Bare Metal

Component	Value/Version	Notes
CPU	Intel x86	The tested CPU is Intel Xeon Gold 6212U 24-core @2.4 GHz
Host OS	Ubuntu	Version 22.04
Kernel Version	Recommended Linux 5.10.x or higher	The tested kernel version is 5.15.x
NIC	• Intel XL710 with Firmware 9.20 0x8000e0e9 0.0.0	
Drivers	virtio	
	i40e version 2.22.18.1	
Kubernetes (K8s)	Version 1.22.x, 1.23.x, 1.25x	The tested K8s version is 1.22.4. K8s version 1.22.2 also works. Cloud-Native Router supports an all-in-one or multinode Kubernetes cluster, with control plane and worker nodes running on virtual machines (VMs) or bare metal servers (BMS).
Calico	Version 3.22.x	
Multus	Version 3.8	
Helm	3.9.x	

Table 2: Minimum Host System Requirements (eBPF) for Bare Metal *(Continued)*

Component	Value/Version	Notes
Container-RT	containerd 1.7.x	Other container runtimes may work but have not been tested with JCNR.
NOTE: The component versions listed in this table are expected to work with JCNR, but not every version or combination is tested in every release.		

NOTE: Cloud-Native Router eBPF XDP Datapath is a *Juniper Technology Preview (Tech Preview)* feature. Limited features are supported. Please review "Juniper Cloud-Native Router vRouter Datapath" on page 11 for more details.

Resource Requirements for Bare Metal

Table 3 on page 40 lists the resource requirements for installing Cloud-Native Router on bare metal servers.

Table 3: Resource Requirements for Bare Metal

Resource	Value	Usage Notes
Data plane forwarding cores	1 core (1P + 1S)	
Service/Control Cores	0	

UIO Driver	VFIO-PCI	To enable, follow the steps below:
		`cat /etc/modules-load.d/vfio.conf`
		`vfio`
		`vfio-pci`

Hugepages (1G)	6 Gi	See "Configure the Number of Huge Pages Available on a Node" on page 401.

Table 3: Resource Requirements for Bare Metal *(Continued)*

Resource	Value	Usage Notes
Cloud-Native Router Controller cores	.5	
Cloud-Native Router vRouter Agent cores	.5	

Miscellaneous Requirements for Bare Metal

Table 4 on page 41 lists additional requirements for installing Cloud-Native Router on bare metal servers.

Table 4: Miscellaneous Requirements for Bare Metal

Requirement	Example
Enable the host with SR-IOV and VT-d in the system's BIOS.	Depends on BIOS.

Enable VLAN driver at system boot.	Configure **/etc/modules-load.d/vlan.conf** as follows:
	`cat /etc/modules-load.d/vlan.conf`
	`8021q`
	Reboot and verify by executing the command:
	`lsmod | grep 8021q`

Table 4: Miscellaneous Requirements for Bare Metal *(Continued)*

Requirement	Example

Enable VFIO-PCI driver at system boot.	Configure **/etc/modules-load.d/vfio.conf** as follows:
	`cat /etc/modules-load.d/vfio.conf`
	`vfio`
	`vfio-pci`
	Reboot and verify by executing the command:
	`lsmod | grep vfio`

Set IOMMU and IOMMU-PT in GRUB.	Add the following line to **/etc/default/grub**.
	`GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
	Update grub and reboot.
	`grub2-mkconfig -o /boot/grub2/grub.cfg`
	`reboot`

Disable spoofcheck on VFs allocated to JCNR. **NOTE**: Applicable for L2 deployments only.	`ip link set <interfacename> vf 1 spoofcheck off.`
Set trust on VFs allocated to JCNR. **NOTE**: Applicable for L2 deployments only.	`ip link set <interfacename> vf 1 trust on`

Table 4: Miscellaneous Requirements for Bare Metal ***(Continued)***

Requirement	Example

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.
 	
NOTE: Applicable for L3 deployments only.
	Create a **/etc/modules-load.d/crpd.conf** file and add the following kernel modules to it:
	
	`tun`
	`fou`
	`fou6`
	`ipip`
	`ip_tunnel`
	`ip6_tunnel`
	`mpls_gso`
	`mpls_router`
	`mpls_iptunnel`
	`vrf`
	`vxlan`

Enable kernel-based forwarding on the Linux host.	`ip fou add port 6635 ipproto 137`

Table 4: Miscellaneous Requirements for Bare Metal *(Continued)*

Requirement	Example

Exclude Cloud-Native Router interfaces from NetworkManager control.	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 Kubernetes management and create problems.
	
	To avoid NetworkManager from interfering with Cloud-Native Router interface configuration, exclude Cloud-Native Router interfaces from NetworkManager control. Here's an example on how to do this in some Linux distributions:
	
	1. Create the **/etc/NetworkManager/conf.d/crpd.conf** file and list the interfaces that you don't want NetworkManager to manage.
	For example:
	`[keyfile]`
	` unmanaged-devices+=interface-name:enp*;interface-name:ens*`
	where `enp*` and `ens*` refer to your Cloud-Native Router interfaces.
	NOTE: `enp*` indicates all interfaces starting with `enp`. For specific interface names, provided a comma-separated list.
	
	2. Restart the NetworkManager service:
	`sudo systemctl restart NetworkManager`
	.
	
	3. Edit the **/etc/sysctl.conf** 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`

Table 4: Miscellaneous Requirements for Bare Metal *(Continued)*

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td></td><td><pre>net.ipv6.conf.default.autoconf=0
net.ipv6.conf.all.autoconf=0</pre>
4. Run the command <code>sysctl -p /etc/sysctl.conf</code> to load the new sysctl.conf values on the host.
5. Create the bond interface manually. For example:
<pre>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</pre></td></tr>
<tr><td>Verify the core_pattern value is set on the host before deploying JCNR.</td><td><pre>sysctl kernel.core_pattern
kernel.core_pattern = |/usr/lib/systemd/systemd-
coredump %P %u %g %s %t %c %h %e</pre>
You can update the core_pattern in <code>/etc/sysctl.conf</code>. For example:
<pre>kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_
%t.gz</pre></td></tr>
<tr><td>Set <code>MACAddressPolicy</code> (Ubuntu only).</td><td>On all nodes running DPDK on a Ubuntu OS, set <code>MACAddressPolicy</code> to <code>none</code> and reboot. For example:
<pre>sudo sed -i 's/MACAddressPolicy=.*/
MACAddressPolicy=none/' /usr/lib/systemd/network/99-
default.link</pre>
<pre>sudo reboot</pre></td></tr>
</table>

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 5: 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 the Cloud-Native Router vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from Cloud-Native Router control plane
TCP	8075, 8076	Telemetry Information- Used for gNMI requests
TCP	9091	vRouter health check–cloud-native router checks to ensure the vRouter agent is running.
TCP	9092	vRouter health check–cloud-native router checks to ensure the vRouter DPDK is running.
TCP	50052	gRPC port–Cloud-Native Router listens on both IPv4 and IPv6
TCP	8081	Cloud-Native Router Deployer Port
TCP	24	cRPD SSH
TCP	830	cRPD NETCONF
TCP	666	**rpd**

Table 5: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	1883	Mosquito mqtt–Publish/subscribe messaging utility
TCP	9500	**agentd** on cRPD
TCP	21883	**na-mqttd**
TCP	50053	Default gNMI port that listens to the client subscription request
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

See "Cloud-Native Router Software Download Packages" on page 387.

Cloud-Native Router Licensing

See "Manage Cloud-Native Router Licenses" on page 352.

Customize Cloud-Native Router Helm Chart for Bare Metal Servers

IN THIS SECTION

Read this topic to learn about the deployment configuration available for the Juniper Cloud-Native Router on bare metal servers.

You can deploy and operate Juniper Cloud-Native Router in the L2, L3, or L2-L3 mode on a bare metal server. You configure the deployment mode by editing the appropriate attributes in the `values.yaml` file prior to deployment.

NOTE:

- 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.

Helm Chart Description for Bare Metal Deployment

Customize the Helm chart using the **Juniper_Cloud_Native_Router_*<release>*/helmchart/jcnr/values.yaml** file. We provide a copy of the default **values.yaml** in "Cloud-Native Router Default Helm Chart" on page 389.

Table 6 on page 49 contains a description of the configurable attributes in **values.yaml** for a bare metal deployment.

Table 6: Helm Chart Description for Bare Metal Deployment

Key	Description
global	
installSyslog	Set to true to install syslog-ng.
registry	Defines the Docker registry for the Cloud-Native Router container images. The default value is `enterprise-hub.juniper.net`. The images provided in the tarball are tagged with the default registry name. If you choose to host the container images to a private registry, replace the default value with your registry URL.
repository	(Optional) Defines the repository path for the Cloud-Native Router container images. This is a global key that takes precedence over the `repository` paths under the `common` section. Default is `jcnr-container-prod/`.
imagePullSecret	(Optional) Defines the Docker registry authentication credentials. You can configure credentials to either the Juniper Networks enterprise-hub.juniper.net registry or your private registry.
registryCredentials	Base64 representation of your Docker registry credentials. See "Configure Repository Credentials" on page 398 for more information.
secretName	Name of the secret object that will be created.
common	Defines repository paths and tags for the various Cloud-Native Router container images. Use defaults unless using a private registry.
repository	Defines the repository path. The default value is `jcnr-container-prod/`. 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.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description
readinessCheck	Set to true to enable Cloud-Native Router Readiness preflight and postflight checks during installation. Comment this out or set to false to disable Cloud-Native Router Readiness preflight and postflight checks. Preflight checks verify that your infrastructure can support JCNR. Preflight checks take place before Cloud-Native Router is installed. Postflight checks verify that your Cloud-Native Router installation is working properly. Postflight checks take place after Cloud-Native Router is installed. See "Cloud-Native Router Readiness Checks" on page 362.
replicas	(Optional) Indicates the number of replicas for cRPD. Default is 1. The value for this key must be specified for multi-node clusters. The value is equal to the number of nodes running JCNR.
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 to L2 and L2-L3 deployments. Enabling this key provides the functionality on all access interfaces. To enable the functionality on trunk interfaces, configure `no-local-switching` in `fabricInterface`. See *Prevent Local Switching* for more details.
iamRole	Not applicable.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>fabricInterface</td><td>
Aggregated interfaces that receive traffic from multiple interfaces. Fabric interfaces are always physical interfaces. They can either be a physical function (PF) or a virtual function (VF). The throughput requirement for these interfaces is higher — hence multiple hardware queues are allocated to them. Each hardware queue is allocated with a dedicated CPU core. See "Cloud-Native Router Interfaces Overview" on page 14 for more information.
Use this field to provide a list of fabric 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 the 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 <code>interface_mode</code> key configured, then the mode of deployment is L2.
When one or more interfaces have an <code>interface_mode</code> key configured along with the rest of the interfaces not having the <code>interface_mode</code> key, then the mode of deployment is L2-L3.
When none of the interfaces have the <code>interface_mode</code> key configured, then the mode of deployment is L3.

For example:
<pre><code># 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

L3 only
- eth1:</code></pre>
</td></tr>
</table>

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description

	`ddp: "off"`
	`qosSchedulerProfileName: "sched_profile_1"`
	
	`# L2L3`
	`- eth1:`
	`ddp: "auto"`
	`qosSchedulerProfileName: "sched_profile_1"`
	`- 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 to 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 in a multi-node cluster.

ddp	(Optional) Indicates the interface-level Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at the 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. See *Enabling Dynamic Device Personalization (DDP) on Individual Interfaces* for more details. Options include auto, on, or off. Default is off. **NOTE**: The interface level `ddp` takes precedence over the global `ddp` configuration.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	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 the desired storm control profile to 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`
	
	See *Native VLAN* for more details.

no-local-switching	Prevents interfaces from communicating directly with each other when configured. Allowed values are true or false. See *Prevent Local Switching* for more details.
qosSchedulerProfileName	Specifies the QoS scheduler profile applicable to this interface. See the `qosSchedulerProfiles` section. If you don't specify a profile, then the QoS scheduler is disabled for this interface, which means that packets are scheduled with no regard to traffic class.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description
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 it set to the default INFO unless instructed to change it by Juniper Networks support.
log_path	The defined directory stores various JCNR-related descriptive logs such as **contrail-vrouter-agent.log**, **contrail-vrouter-dpdk.log**, etc. Default is **/var/log/jcnr/**.
syslog_notifications	Indicates the absolute path to the file that stores `syslog-ng` generated notifications in JSON format. Default is **/var/log/jcnr/jcnr_notifications.json**.
corePattern	Indicates the `core_pattern` for the core file. If left blank, then Cloud-Native Router pods will not overwrite the default pattern on the host. **NOTE**: Set the `core_pattern` on the host before deploying JCNR. You can 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 to the `core` file. Default is **/var/crash**.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description

nodeAffinity	(Optional) Defines labels on nodes to determine where to place the vRouter pods.
	
	By default the vRouter pods are deployed to all nodes of a cluster.
	
	In the example below, the node affinity label is defined as `key1=jcnr`. You must apply this label to each node where Cloud-Native Router is to be deployed:
	
	`nodeAffinity:`
	`- key: key1`
	`operator: In`
	`values:`
	`- jcnr`
	
	NOTE: This key is a global setting.

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 path with the path in your distribution (for example, **/var/opt/cni/bin**).
grpcTelemetryPort	(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50053.
grpcVrouterPort	(Optional) Default is 50052. Configure to override.
vRouterDeployerPort	(Optional) Default is 8081. Configure to override.
jcnr-vrouter	

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description
cpu_core_mask	If present, this indicates that you want to use static CPU allocation to allocate cores to the forwarding plane. This value should be a comma-delimited list of isolated CPU cores that you want to statically allocate to the forwarding plane (for example, `cpu_core_mask: "2,3,22,23"`). Use the cores not used by the host OS. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Cloud-Native Router Readiness preflight checks, if enabled, will fail the installation if you specify both.
guaranteedVrouterCpus	If present, this indicates that you want to use the Kubernetes CPU Manager to allocate CPU cores to the forwarding plane. This value should be the number of guaranteed CPU cores that you want the Kubernetes CPU Manager to allocate to the forwarding plane. You should set this value to at least one more than the number of forwarding cores. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Using both can lead to unpredictable behavior.
dpdkCtrlThreadMask	Specifies the CPU core(s) to allocate to vRouter DPDK control threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `serviceCoreMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description
serviceCoreMask	Specifies the CPU core(s) to allocate to vRouter DPDK service threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `dpdkCtrlThreadMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.
numServiceCtrlThreadCPU	Specifies the number of CPU cores to allocate to vRouter DPDK service/control traffic when using the Kubernetes CPU Manager. This number should be smaller than the number of `guaranteedVrouterCpus` cores. The remaining `guaranteedVrouterCpus` cores are allocated for forwarding. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
numberOfSchedulerLcores	The number of CPU cores that you want Kubernetes CPU Manager to dedicate to your QoS schedulers. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
restoreInterfaces	Set to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts or if Cloud-Native Router is uninstalled.
bondInterfaceConfigs	(Optional) Enable bond interface configurations only for L2 or L2-L3 deployments.
name	Name of the bond interface.
mode	Set to 1 (active-backup).
slaveInterfaces	List of fabric interfaces to be bonded.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description
primaryInterface	(Optional) Primary interface for the bond.
slaveNetworkDetails	Not applicable.
mtu	Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default is 9000.
qosSchedulerProfiles	Defines the QoS scheduler profiles that are referenced from the `fabricInterface` section.
sched_profile_1	The name of the QoS scheduler profile.
	cpu Specify the CPU core(s) to dedicate to the scheduler. If `cpu_core_mask` is specified, this should be a unique additional core(s). **bandwidth** Specify the bandwidth in Gbps.
stormControlProfiles	Configure the rate limit profiles for BUM traffic on fabric interfaces in bytes per second. See */Content/l2-bum-rate-limiting_xi931744_1_1.dita* for more details.
dpdkCommandAdditionalArgs	Pass any additional DPDK command line parameters. The `--yield_option 0` is set by default and implies the DPDK forwarding cores will not yield their assigned CPU cores. Other common parameters that can be added are `tx` and `rx` descriptors and `mempool`. For example: `dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 2048 --dpdk_rxd_sz 2048 --vr_mempool_sz 131072"` **NOTE**: Changing the number of `tx` and `rx` descriptors and the `mempool` size affects the number of huge pages required. If you make explicit changes to these parameters, set the number of huge pages to 10 (x 1 GB). See "Configure Huge Pages" on page 401 for information on how to configure huge pages.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description
dpdk_monitoring_thread_config	(Optional) Enables a monitoring thread for the vRouter DPDK container. Every `loggingInterval` seconds, a log containing the information indicated by `loggingMask` is generated.
loggingMask	Specifies the information to be generated. Represented by a bitmask with bit positions as follows: • 0b001 is the nl_counter • 0b010 is the lcore_timestamp • 0b100 is the profile_histogram
loggingInterval	Specifies the log generation interval in seconds.
ddp	(Optional) Indicates the global Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at the 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. See *Enabling Dynamic Device Personalization (DDP) on Individual Interfaces* for more details. Options include auto, on, or off. Default is off. **NOTE**: The interface level `ddp` takes precedence over the global `ddp` configuration.
twampPort	(Optional) The TWAMP session reflector port (if you want TWAMP sessions to use vRouter timestamps). The vRouter listens to TWAMP test messages on this port and inserts/ overwrites timestamps in TWAMP test messages. Timestamping TWAMP messages at the vRouter (instead of at cRPD) leads to more accurate measurements. Valid values are 862 and 49152 through 65535. If this parameter is absent, then the vRouter does not insert or overwrite timestamps in the TWAMP session. Timestamps are taken and inserted by cRPD instead. See *Two-Way Active Measurement Protocol (TWAMP)*.

Table 6: Helm Chart Description for Bare Metal Deployment *(Continued)*

Key	Description
vrouter_dpdk_uio_driver	The uio driver is `vfio-pci`.
agentModeType	Options are dpdk or xdp. Set to dpdk to bring up the DPDK datapath. Set to xdp to use eBPF. Default is dpdk. Note: xdp is supported for bare metal deployments only. See "Juniper Cloud-Native Router vRouter Datapath" on page 11 for more details.
fabricRpfCheckDisable	Set to false to enable the RPF check on all Cloud-Native Router fabric interfaces. By default, RPF check is disabled.
telemetry	(Optional) Configures cRPD telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
disable	Set to true to disable cRPD telemetry. Default is false, which means that cRPD telemetry is enabled by default.
metricsPort	The port that the cRPD telemetry exporter is listening to Prometheus queries on. Default is 8072.
logLevel	One of warn, warning, info, debug, trace, or verbose. Default is info.
gnmi	(Optional) Configures cRPD gNMI settings.
	enable Set to true to enable the cRPD telemetry exporter to respond to gNMI requests.
vrouter	
telemetry	(Optional) Configures vRouter telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .

Table 6: Helm Chart Description for Bare Metal Deployment ***(Continued)***

Key	Description
	metricsPort Specify the port that the vRouter telemetry exporter listens to Prometheus queries on. Default is 8070.
	logLevel One of warn, warning, info, debug, trace, or verbose. Default is info.
	gnmi (Optional) Configures vRouter gNMI settings. enable - Set to true to enable the vRouter telemetry exporter to respond to gNMI requests.
persistConfig	Set to true if you want Cloud-Native Router pod configuration to persist even after uninstallation. This option can only be set for L2 mode deployments. Default is false.
enableLocalPersistence	Set to true if you're using the cRPD CLI or NETCONF to configure JCNR. When set to true, the cRPD CLI and NETCONF configuration persists through node reboots, cRPD pod restarts, and Cloud-Native Router upgrades. Default is false.
interfaceBoundType	Not applicable.
networkDetails	Not applicable.
networkResources	Not applicable.
contrail-tools	
install	Set to true to install contrail-tools (used for debugging).

Customize Cloud-Native Router Configuration

SUMMARY

Read this topic to understand how to customize Cloud-Native Router configuration using a Configlet custom resource.

IN THIS SECTION

Configlet Custom Resource

Starting with Juniper Cloud-Native Router (JCNR) Release 24.2, we support customizing Cloud-Native Router configuration using a configlet custom resource. The configlet can be generated either by rendering a predefined template of supported Junos configuration or using raw configuration. The generated configuration is validated and deployed on the Cloud-Native Router controller (cRPD) as one or more Junos configuration groups.

NOTE: You can configure Cloud-Native Router using either configlets or the cRPD CLI or NETCONF. If you use the cRPD CLI or NETCONF, be sure to enable local persistence in **values.yaml** (`enableLocalPersistence: true`) so that your CLI or NETCONF configuration persists across reboots and upgrades.

NOTE: Using both configlets and the cRPD CLI or NETCONF to configure Cloud-Native Router may lead to unpredictable behavior. Use one or the other, but not both.

Configuration Examples

You create a configlet custom resource of the kind `Configlet` in the `jcnr` namespace. You provide raw configuration as Junos `set` commands.

Use `crpdSelector` to control where the configlet applies. The generated configuration is deployed to cRPD pods on nodes matching the specified label only. If `crpdSelector` is not defined, the configuration is applied to all cRPD pods in the cluster.

An example configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample             # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker                    # <-- Node label to select the cRPD pods
```

You can also use a templatized configlet yaml that contains keys or variables. The values for variables are provided by a configletDataValue custom resource, referenced by `configletDataValueRef` . An example templatized configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-with-template      # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: worker                           # <-- Node label to select the cRPD pods
  configletDataValueRef:
    name: "configletdatavalue-sample"        # <-- Configlet Data Value resource name
```

To render configuration using the template, you must provide key:value pairs in the ConfigletDataValue custom resource:

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
```

```
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"           # <-- Key:Value pair
  }
```


The generated configuration is validated and applied to all or selected cRPD pods as a Junos Configuration Group.

Applying the Configlet Resource

The configlet resource can be used to apply configuration to selected or all cRPD pods either when Cloud-Native Router is deployed or once the cRPD pods are up and running. Let us look at configlet deployment in detail.

Applying raw configuration

1. Create raw configuration configlet yaml. The example below configures a loopback interface in cRPD.

```
cat configlet-sample.yaml
```


```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker
```

2. Apply the configuration using the `kubectl apply` command.

   ```
   kubectl apply -f configlet-sample.yaml
   ```

   ```
   configlet.configplane.juniper.net/configlet-sample created
   ```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

   ```
   kubectl get nodeconfiglets -n jcnr
   ```

   ```
   NAME                              AGE
   configlet-sample-node1            10m
   ```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

   ```
   kubectl describe nodeconfiglet configlet-sample-node1 -n jcnr
   ```

   The following output has been trimmed for brevity:

   ```
   Name:         configlet-sample-node1
   Namespace:    jcnr
   Labels:       core.juniper.net/nodeName=node1
   Annotations:  <none>
   API Version:  configplane.juniper.net/v1
   Kind:         NodeConfiglet
   Metadata:
     Creation Timestamp:  2024-06-13T16:51:23Z
    ...
   Spec:
     Clis:
       set interfaces lo0 unit 0 address 10.10.10.1/32
     Group Name:  configlet-sample
   ```

```
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 10.10.10.1/32;
            }
        }
    }
}
```

NOTE: The configuration generated using configlets is applied to cRPD as configuration groups. We therefore recommend that you not use configuration groups when specifying your configlet.

Applying templatized configuration

1. Create the templatized configlet yaml and the configlet data value yaml for key:value pairs.

```
cat configlet-sample-template.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-template
  namespace: jcnr
```

```
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: master
  configletDataValueRef:
    name: "configletdatavalue-sample"
```

```
cat configletdatavalue-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"
  }
```

2. Apply the configuration using the `kubectl apply` command, starting with the config data value yaml.

```
kubectl apply -f configletdatavalue-sample.yaml
```

```
configletdatavalue.configplane.juniper.net/configletdatavalue-sample created
```

```
kubectl apply -f configlet-sample-template.yaml
```

```
configlet.configplane.juniper.net/configlet-sample-template created
```

3. Check on the configlet.

When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                                       AGE
configlet-sample-template-node1            10m
```

If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

For example:

```
kubectl describe nodeconfiglet configlet-sample-template-node1 -n jcnr
```

The following output has been trimmed for brevity:

```
Name:         configlet-sample-template-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
Annotations:  <none>
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample-template
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample-template
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32;
            }
        }
    }
}
```

Modifying the Configlet

You can modify a configlet resource by changing the yaml file and reapplying it using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample configured
```

Any changes to existing configlet resource are reconciled by replacing the configuration group on cRPD.

You can delete the configuration group by deleting the configlet resource using the `kubectl delete` command.

```
kubectl delete configlet configlet-sample -n jcnr
```

```
configlet.configplane.juniper.net "configlet-sample" deleted
```

Troubleshooting

If you run into problems, check the `contrail-k8s-deployer` logs. For example:

```
kubectl logs contrail-k8s-deployer-8ff895cc5-cbfwm -n contrail-deploy
```

Cloud-Native Router Operator Service Module: Host-Based Routing

SUMMARY

The Cloud-Native Router Operator Service Module is an operator framework that we use to develop cRPD applications and solutions. This section describes how to use the Service Module to implement a host-based routing solution for your Kubernetes cluster.

IN THIS SECTION

Overview

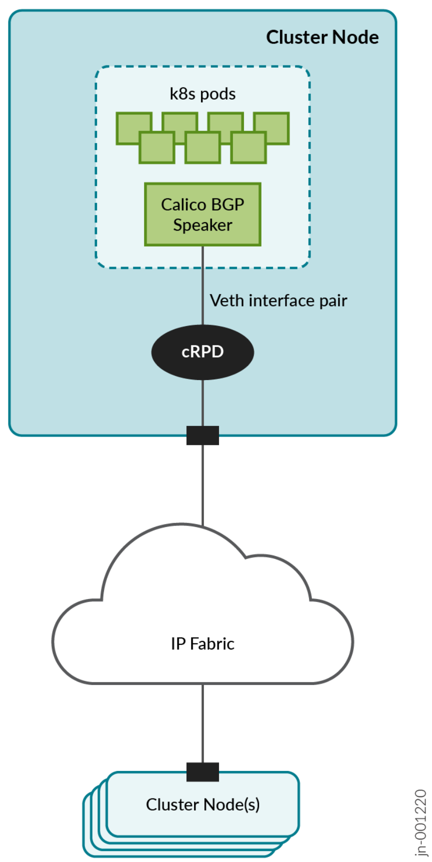
We provide the Cloud-Native Router Operator Service Module to implement a cRPD host-based routing solution for your Kubernetes cluster. Host-based routing, also known as host-based networking, refers to using the host's network namespace instead of the Kubernetes default namespace for the pod network. In the context of Cloud-Native Router, this means that pod-to-pod traffic traverses an external (to the cluster) network provided by cRPD.

The Kubernetes CNI exposes container network endpoints through BGP to a co-located (but independent) cRPD instance acting as a BGP peer. Packets between pods are routed by the Kubernetes CNI to this cRPD instance. This cRPD instance, in turn, routes the packets to the cRPD instance on the destination node for hand-off to the destination Kubernetes CNI for delivery to the destination pod.

By taking this approach to Kubernetes host networking, we are leveraging Cloud-Native Router to provide a more fulsome pod networking implementation that supports common data center protocols such as EVPN-VXLAN and MPLS over UDP.

Figure 3 on page 71 shows a Kubernetes cluster leveraging cRPD for host-based routing. The Calico BGP speaker in the cluster connects through a virtual Ethernet (veth) interface to a co-located cRPD instance attached to the IP fabric interconnecting cluster nodes.

Figure 3: Host-Based Routing



To facilitate the installation of this host-based routing solution, we provide a Helm chart that you can install and we show you how to configure and customize cRPD and the underlying network infrastructure to support a 5-node cluster (3 control plane nodes and 2 worker nodes).

Install Host-Based Routing

This is the main procedure. Start here.

1. "Prepare the Nodes" on page 76.
2. "Create Virtual Ethernet Interface (VETH) Pairs and Configure Static Routes" on page 78.
3. Pick one of the control plane nodes as the installation host and install Helm on it. The installation host is where you'll install the Helm chart.

```
curl -sL https://get.helm.sh/helm-v3.9.4-linux-amd64.tar.gz
```

```
tar -xvzf helm-v3.9.4-linux-amd64.tar.gz
```

```
sudo mv linux-amd64/helm /usr/local/bin/helm
```

```
rm -rf linux-amd64
```

4. Install cRPD on all nodes.

   For convenience, we provide an example script ("Example cRPD Installation Script" on page 418) that installs cRPD on a node. Run the script with the respective configuration file on each node.

```
./install-crpd.sh
```

   For more information on how to install cRPD, see https://www.juniper.net/documentation/us/en/software/crpd/crpd-deployment/index.html.

5. Verify that the `veth-crpd` interface is reachable from the local host.

   For example:

```
ping 10.1.1.2
```

6. Verify that BGP sessions are established between cRPDs and check the routing table.

   a. Exec into the cRPD container on the local node.

```
sudo podman exec -it <crpd-container> bash
```

where *<crpd-container>* is the name of the cRPD container running on the local node.

b. Enter CLI mode.

```
cli
```

c. Check that BGP sessions have been established.

```
show bgp summary
```

If you're on a control plane node, then you'll see BGP sessions established between the local cRPD instance and the cRPD instances on all other nodes. If you're on a worker node, then you'll see BGP sessions established between the local cRPD instance and the cRPD instances on all the control plane nodes.

d. Check that `veth-k8s` routes are in the routing table.

```
show route table master-calico-ri.inet.0
```

Make sure that the `veth-k8s` routes (for example, 10.1.1.1, 10.1.2.1, 10.1.3.1, 10.1.4.1, 10.1.5.1) are in the routing table.

7. Verify that `veth-k8s` interfaces are reachable from each node to all other nodes.

   For example:

```
ping 10.1.5.1
```

8. Configure the kubelet on all nodes to use the local `veth-k8s` IPv4 address as the node IP.

```
echo "KUBELET_EXTRA_ARGS=--node-ip=<veth-k8s-ip>" | sudo tee /etc/default/kubelet > /dev/
null
```

where *<veth-k8s-ip>* is the `<veth-k8s>` IPv4 address as shown in Table 7 on page 78 (minus the /30 subnet qualifier).

```
sudo systemctl restart kubelet
```

Perform this step on all nodes, but use the respective `<veth-k8s>` IPv4 addresses.

9. Create the first control plane node in the Kubernetes cluster.

Log in to one of the control plane nodes and create the cluster.

```
sudo kubeadm init --pod-network-cidr=<pod-cidr> --apiserver-advertise-address=<veth-k8s-ip>
--control-plane-endpoint=<veth-k8s-ip> --upload-certs
```

where *<pod-cidr>* is `192.168.0.0/24` in our example (or if running dual stack `192.168.0.0/24,2001:db8:42:0::56`),

and *<veth-k8s-ip>* is the `<veth-k8s>` IPv4 address of the local control plane node.

10. Log in to each of the other two control plane nodes and join each to the cluster.

    For example:

    ```
    kubeadm join 10.1.1.1:6443 <options> --control-plane
    ```

11. Log in to each of the two worker nodes and join each to the cluster.

    For example:

    ```
    kubeadm join 10.1.1.1:6443 <options>
    ```

12. Verify that all nodes are now part of the cluster.

    ```
    kubectl get nodes
    ```

13. Untaint all control plane nodes so that all pods can run on them.

    ```
    kubectl taint nodes --all node-role.kubernetes.io/control-plane-
    ```

14. Install Calico.

    See https://docs.tigera.io/calico/latest/getting-started/kubernetes/quickstart#install-calico.

15. Configure Calico.

    a. Disable nodeMesh and set the AS number and listen port.

    ```
    kubectl apply -f bgpconfig.yaml
    ```

    See "BGP Configuration Example" on page 436.

b. Configure the IPv4 address pool with no IP-IP or VxLAN encapsulation.

```
kubectl apply -f ippool-v4.yaml
```

See "IP Pool Configuration Example" on page 436.

c. (Optional) If you're running a dual stack setup, then configure the IPv6 address pool.

```
kubectl apply -f ippool-v6.yaml
```

See "IP Pool Configuration Example" on page 436.

d. Configure the IPv4 BGP peering relationships.

```
kubectl apply -f bgppeers-v4.yaml
```

See "BGP Peer Configuration Example" on page 437.

e. (Optional) If you're running a dual stack setup, then configure the IPv6 BGP peering relationships.

```
kubectl apply -f bgppeers-v6.yaml
```

See "BGP Peer Configuration Example" on page 437.

16. Verify that BGP sessions are established between the Calico CNI and the co-located cRPD.

a. Exec into the cRPD container on the local node.

```
sudo podman exec -it <crpd-container> bash
```

where *<crpd-container>* is the name of the cRPD container running on the local node.

b. Enter CLI mode.

```
cli
```

c. Check that BGP sessions have been established.

```
show bgp summary
```

In addition to the BGP sessions that you saw earlier between cRPDs, you'll see a BGP session established between the local cRPD and the Calico CNI.

17. "Install the Operator Service Module" on page 81.

18. (Optional) If you want to set up a secondary CNI that also uses cRPD, then see "Set Up Secondary CNI for Host-Based Routing" on page 83.

Prepare the Nodes

Perform the following steps on all the nodes (VMs or bare metal servers) that you want to be in your cluster. All nodes should have at least 2 interfaces:

- one interface for regular management access (for example, SSH)
- one interface for cRPD to connect to the IP fabric

1. Install a fresh OS.

   We tested our host-based routing solution on the following combination:

   - Ubuntu 22.04
   - Linux kernel 5.15.0-88-generic

2. Update the repository list and install podman.

   ```
   sudo apt update
   ```

   ```
   sudo install -y podman
   ```

3. Install the required kernel modules on all nodes in the cluster.

   Create **/etc/modules-load.d/jcnr.conf** and populate it with the following list of kernel modules:

   ```
   overlay
   br_netfilter
   8021q
   uio_pci_generic
   vfio-pci
   tun
   fou
   fou6
   ipip
   ip_tunnel
   ```

```
ip6_tunnel
mpls_gso
mpls_router
mpls_iptunnel
vrf
vxlan
```

4. Enable IP forwarding and iptables on the underlay Linux bridges.

   Create **/etc/sysctl.d/99-kubernetes-cri.conf** and populate it with the following configuration:

```
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv6.conf.all.forwarding = 1
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
```

   Additionally, set the following in **/etc/sysctl.conf**:

```
net.ipv4.ip_forward = 1
net.ipv6.conf.all.forwarding = 1
```

   NOTE: The above enables IP forwarding and iptables for both IPv4 and IPv6. If you're only running IPv4, then omit the IPv6 settings.

5. Set the MAC address policy.

```
sudo sed -i 's/MACAddressPolicy=.*/MACAddressPolicy=none/' /usr/lib/systemd/network/99-
default.link
```

6. Install kubeadm, kubelet, and kubectl. See https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/.

   We tested our host-based routing solution on Kubernetes version 1.28.15.

7. Reboot.

Create Virtual Ethernet Interface (VETH) Pairs and Configure Static Routes

Before we bring up the Kubenetes cluster and cRPD, we'll create the veth interfaces that connect them together and configure static routes to direct traffic to the cRPD instance.

On each node, we'll run the Kubernetes cluster (including the Calico BGP speaker) in the Kubernetes default namespace and we'll run cRPD in a namespace that we'll call `crpd`. We'll create a veth pair that connects the two namespaces, from `veth-k8s` in the default namespace to `veth-crpd` in the `crpd` namespace.

This is shown in Table 7 on page 78 along with the IP address assignments we'll use in our example. This includes both IPv4 and IPv6 addresses for a dual stack deployment. If you're only running IPv4, then ignore the IPv6 settings.

Table 7: Namespace and Interface Configuration (Example)

Node	Namespace	Interface	IP Address
Node 1 (control plane)	default	veth-k8s	10.1.1.1/30 2001:db8:1::1/126
	crpd	veth-crpd	10.1.1.2/30 2001:db8:1::2/126
		ens4[1]	192.168.1.101/24
Node 2 (control plane)	default	veth-k8s	10.1.2.1/30 2001:db8:2::1/126
	crpd	veth-crpd	10.1.2.2/30 2001:db8:2::2/126
		ens4[1]	192.168.1.102/24
Node 3 (control plane)	default	veth-k8s	10.1.3.1/30 2001:db8:3::1/126

Table 7: Namespace and Interface Configuration (Example) ***(Continued)***

Node	Namespace	Interface	IP Address
	crpd	veth-crpd	10.1.3.2/30 2001:db8:2::2/126
		ens4[1]	192.168.1.103/24
Node 4 (worker)	default	veth-k8s	10.1.4.1/30 2001:db8:4::1/126
	crpd	veth-crpd	10.1.4.2/30 2001:db8:4::2/126
		ens4[1]	192.168.1.104
Node 5 (worker)	default	veth-k8s	10.1.5.1/30 2001:db8:5::1/126
	crpd	veth-crpd	10.1.5.2/30 2001:db8:5::2/126
		ens4[1]	192.168.1.105/24
[1] This is the physical underlay interface connecting cRPD to the IP fabric. The interface name in your setup may differ.			

Perform the following steps on all nodes in the cluster. Remember to set the IP addresses for the different nodes as shown in Table 7 on page 78.

1. Create `veth-k8s` and `veth-crpd` and pair them together.

a. Create the veth interface pair.

```
sudo ip link add dev veth-k8s type veth peer name veth-crpd
```

By default, both interfaces are in the default namespace. We'll move `veth-crpd` to the `crpd` namespace in a later step.

b. Enable these 2 veth interfaces.

```
sudo ip link set dev veth-k8s up
```

```
sudo ip link set dev veth-crpd up
```

c. Configure the IP address on the `veth-k8s` interface.

```
sudo ip addr add 10.1.1.1/30 dev veth-k8s
```

We'll configure the IP address for the `veth-crpd` interface in a later step.

d. (Optional) If you want to run a dual IPv4/IPv6 stack setup, then configure the IPv6 address on the same `veth-k8s` interface.

```
sudo ip addr add 2001:db8:1::1/126 dev veth-k8s
```

We'll configure the IPv6 address for the `veth-crpd` interface in a later step.

2. Create the `crpd` namespace for cRPD.

```
sudo ip netns add crpd
```

3. Move `veth-crpd` to the `crpd` namespace.

a. Assign the physical underlay interface to the `crpd` namespace. This is the interface that connects cRPD to the IP fabric.

For example:

```
sudo ip link set ens4 netns crpd
```

where `ens4` is the physical interface (in our example) that connects to the IP fabric.

b. Configure the IP address for the `ens4` interface.

```
sudo ip netns exec crpd ifconfig ens4 192.168.1.101/24 up
```

where `192.168.1.0/24` is the underlay subnet connecting to the IP fabric.

c. Assign the `veth-crpd` interface to the `crpd` namespace.

```
sudo ip link set veth-crpd netns crpd
```

d. Configure the IP address for the `veth-crpd` interface.

```
sudo ip netns exec crpd ifconfig veth-crpd 10.1.1.2/30 up
```

e. (Optional) If you want to run a dual IPv4/IPv6 stack setup, then configure the IPv6 address for the `veth-crpd` interface.

```
sudo ip netns exec crpd ip addr add 2001:db8:1::2/126 dev veth-crpd
```

4. In the default namespace, configure a route to all cRPD interfaces.

```
sudo ip route add 10.1.0.0/16 via 10.1.1.2
```

5. Repeat step 1 through step 4 for Nodes 2 through 5 according to Table 7 on page 78.

Install the Operator Service Module

Run these steps on the nodes indicated. The installation host is the control plane node where you installed Helm earlier.

1. On the installation host, create the `jcnr` namespace.

```
kubectl create ns jcnr
```

2. On the installation host, create and apply the JCNR secret.

Create a **jcnr-secrets.yaml** file with the below contents.

```
apiVersion: v1
kind: Secret
metadata:
  name: jcnr-secrets
  namespace: jcnr
type: Opaque
data:
  root-password: <password>
  crpd-license: <crpd-license>
```

where *<password>* is the base64-encoded string of the root password and *<crpd-license>* is the base64-encoded cRPD license. For more information on installing your cRPD license, see "Installing Your License" on page 353.

Apply the secret.

```
kubectl apply -f jcnr-secrets.yaml
```

3. On all nodes, create **/etc/crpd/crpd_conf.yaml** with the content below.

```
management_ip: <veth-crpd-ip>
```

   where *<veth-crpd-ip>* is the IPv4 address of the `<veth-crpd>` interface on the local node.

4. On the installation host, download the Cloud-Native Router Operator Service Module package.

   You can download the Service Module package from the Juniper Networks software download site. See "Cloud-Native Router Software Download Packages" on page 387.

5. Gunzip and untar the software package.

```
tar -xzvf Juniper_Cloud_Native_Router_Service_Module_<release>.tar.gz
```

6. Load the provided images on all nodes in the cluster. The images are located in the downloaded package.

   See "Deploy Prepackaged Images" on page 399.

7. On the installation host, extract the Cloud-Native Operator Service Module Helm chart.

a. Navigate to the Helm chart directory.

```
cd Juniper_Cloud_Native_Router_Service_Module_<release>/helmchart
```

b. Extract the Helm chart.

```
tar -xzvf jcnr-gwsvc-<release>.tgz
```

8. Apply the Helm chart.

```
cd jcnr-gwsvc
```

```
helm upgrade --install --wait svc . --set
controller.nodeAffinity[0].operator=Exists,controller.nodeAffinity[0].key=node-
role.kubernetes.io/control-
plane,global.hostBasedNetworking=true,replicaCount=3,controller.image.pullPolicy=IfNotPresent,
crd_loader.pullPolicy=IfNotPresent
```

Set the `replicaCount` to the number of control plane nodes in the cluster.

9. After a few minutes, verify that the cluster is up and running.

```
kubectl get pods -A
```

```
helm ls
```

Set Up Secondary CNI for Host-Based Routing

This procedure shows an example of how to set up a secondary MACVLAN CNI and a secondary IPVLAN CNI for host-based routing.

1. On the installation host, install multus.

   See https://github.com/k8snetworkplumbingwg/multus-cni/blob/master/docs/quickstart.md.

2. On all nodes, create the veth interface pairs for MACVLAN.

```
sudo ip link add dev host-end type veth peer name vrf-end
sudo ip link set dev host-end up
sudo ip link set dev vrf-end up
sudo ip link set vrf-end netns crpd
```

where `host-end` is the veth endpoint on the Kubernetes cluster and `vrf-end` is the veth endpoint on cRPD.

3. On all nodes, create the veth interface pairs for IPVLAN.

```
sudo ip link add dev ipvlan-host type veth peer name ipvlan-vrf
sudo ip link set dev ipvlan-host up
sudo ip link set dev ipvlan-vrf up
sudo ip link set ipvlan-vrf netns crpd
```

where `ipvlan-host` is the veth endpoint on the Kubernetes cluster and `ipvlan-vrf` is the veth endpoint on cRPD.

4. For IPVLAN, on all nodes, enable proxy ARP on `ipvlan-vrf`.

```
sudo ip netns exec crpd bash
echo "1" > /proc/sys/net/ipv4/conf/ipvlan-vrf/proxy_arp
exit
```

5. Check the interfaces on all nodes.

```
sudo ip netns exec crpd ifconfig
```

If, for some reason, the interfaces are not up, set them up from cRPD as follows:

```
sudo ip netns exec crpd ip link set dev vrf-end up
```

```
sudo ip netns exec crpd ip link set dev ipvlan-vrf up
```

6. On the installation host, create and apply the default VxLAN and route target pools.

```
kubectl apply -f vxlan-pool.yaml
```

```
kubectl apply -f rt-pool.yaml
```

See "Host-Based Routing: Example VxLAN and Route Target Pools" on page 440.

7. Label all the nodes.

```
kubectl label nodes <cp-nodename> master=""
kubectl label nodes <worker-nodename> worker=""
```

where *<cp-nodename>* and *<worker-nodename>* are the node names of the control plane and worker nodes respectively.

8. Configure JCNR.

```
kubectl create ns hbn
```

```
kubectl apply -f jcnr-config.yaml
```

See "JCNR Configuration" on page 441.

9. Apply the MACVLAN custom resource.

```
kubectl apply -f macvlan-cr.yaml
```

See "Example MACVLAN Custom Resource" on page 442.

10. Create MACVLAN pods.

```
kubectl apply -f macvlan-pods.yaml
```

See "Example MACVLAN Pods" on page 445.

11. Apply the IPVLAN custom resource.

```
kubectl apply -f ipvlan-cr.yaml
```

See "Example IPVLAN Custom Resource" on page 447.

12. Create IPVLAN pods.

```
kubectl apply -f ipvlan-pods.yaml
```

See "Example IPVLAN Pods" on page 450.

3

CHAPTER

Install Cloud-Native Router on Red Hat OpenShift

IN THIS CHAPTER

Install and Verify Juniper Cloud-Native Router for OpenShift Deployment

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 on Red Hat OpenShift Container Platform (OCP).

IN THIS SECTION

Install Juniper Cloud-Native Router Using Helm Chart

Read this section to learn the steps required to install the cloud-native router components using Helm charts.

1. Review the "System Requirements for OpenShift Deployment" on page 98 to ensure the cluster has all the required configuration.
2. Download the desired Cloud-Native Router software package to the directory of your choice.

   You have the option of downloading the package to install Cloud-Native Router only or downloading the package to install JNCR together with Juniper cSRX. See "Cloud-Native Router Software Download Packages" on page 387 for a description of the packages available. If you don't want to install Juniper cSRX now, you can always choose to install Juniper cSRX on your working Cloud-Native Router installation later.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

   ```
   tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
   ```
4. Change directory to the main installation directory.

- If you're installing Cloud-Native Router only, then:

  ```
  cd Juniper_Cloud_Native_Router_<release>
  ```

  This directory contains the Helm chart for Cloud-Native Router only.

- If you're installing Cloud-Native Router and cSRX at the same time, then:

  ```
  cd Juniper_Cloud_Native_Router_CSRX_<release>
  ```

  This directory contains the combination Helm chart for Cloud-Native Router and cSRX.

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

5. View the contents in the current directory.

   ```
   ls
   helmchart  images  README.md  scripts secrets
   ```

6. Change to the **helmchart** directory and expand the Helm chart.

   ```
   cd helmchart
   ```

   - For Cloud-Native Router only:

     ```
     ls
     jcnr-<release>.tgz
     ```

     ```
     tar -xzvf jcnr-<release>.tgz
     ```

     ```
     ls
     jcnr   jcnr-<release>.tgz
     ```

     The Helm chart is located in the **jcnr** directory.

- For the combined Cloud-Native Router and cSRX:

```
ls
jcnr_csrx-<release>.tgz
```

```
tar -xzvf jcnr_csrx-<release>.tgz
```

```
ls
jcnr_csrx   jcnr_csrx-<release>.tgz
```

  The Helm chart is located in the **jcnr_csrx** directory.

7. 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 398 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 399 for instructions on how to import images to the local container runtime.

8. Follow the steps in "Installing Your License" on page 353 to install your Cloud-Native Router license.

9. 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**.

10. Apply **secrets/jcnr-secrets.yaml** to the cluster.

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

11. If desired, configure how cores are assigned to the vRouter DPDK containers. See "Allocate CPUs to the Cloud-Native Router Forwarding Plane" on page 355.

12. Customize the Helm chart for your deployment using the **helmchart/jcnr/values.yaml** or **helmchart/jcnr_csrx/values.yaml** file.

    See "Customize Cloud-Native Router Helm Chart for OpenShift Deployment" on page 111 for descriptions of the Helm chart configurations.

13. Optionally, customize Cloud-Native Router configuration.

    See "Customize Cloud-Native Router Configuration " on page 62 for creating and applying the cRPD customizations.

14. If you're installing Juniper cSRX now, then follow the procedure in "Apply the cSRX License and Configure cSRX" on page 338.

15. Deploy the Juniper Cloud-Native Router using the Helm chart.

    Navigate to the **helmchart/jcnr** or the **helmchart/jcnr_csrx** directory and run the following command:

```
helm install jcnr .
```

or

```
helm install jcnr-csrx .
```

```
NAME: jcnr
LAST DEPLOYED: Fri Dec 22 06:04:33 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

16. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

Sample output:

```
NAME        NAMESPACE    REVISION       UPDATED
STATUS          CHART                     APP VERSION
jcnr        default      1              2023-12-22 06:04:33.144611017 -0400 EDT
deployed        jcnr-<version>                <version>
```

Verify Installation

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

> **NOTE**: The output shown in this example procedure is affected by the number of nodes in the cluster. The output you see in your setup may differ in that regard.

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                                          jcnr-0-dp-contrail-vrouter-nodes-b2jxp             2/2
   Running            0              16d
   contrail                                          jcnr-0-dp-contrail-vrouter-nodes-vrdpdk-g7wrk      1/1
   Running            0              16d
   jcnr                                              jcnr-0-crpd-0                                      1/1
   Running            0              16d
   jcnr                                              syslog-ng-vh89p                                    1/1
   Running            0              16d
   openshift-cluster-node-tuning-operator            tuned-zccwc                                        1/1
   Running            8              69d
   openshift-dns                                     dns-default-wmchn                                  2/2
   Running            14             69d
   ```

```
openshift-dns                                   node-resolver-dm9b7                                 1/1
Running            8              69d
openshift-image-registry                        image-pruner-28212480-bpn9w                         0/1
Completed          0              2d11h
openshift-image-registry                        image-pruner-28213920-9jk74                         0/1
Completed          0              35h
openshift-image-registry                        node-ca-jbwlx                                       1/1
Running            8              69d
openshift-ingress-canary                        ingress-canary-k6jqs                                1/1
Running            8              69d
openshift-ingress                               router-default-55dff9cbc5-kz8bg                     1/1
Running            1              62d
openshift-kni-infra                             coredns-node-warthog-41                             2/2
Running            16             69d
openshift-kni-infra                             keepalived-node-warthog-41                          2/2
Running            14             69d
openshift-machine-config-operator               machine-config-daemon-w8fbh                         2/2
Running            16             69d
openshift-monitoring                            alertmanager-main-1                                 6/6
Running            7              62d
openshift-monitoring                            node-exporter-rbht9                                 2/2
Running            15             69d
openshift-monitoring                            prometheus-adapter-7d77cfb894-nx29s                 1/1
Running            0              6d18h
openshift-monitoring                            prometheus-k8s-1                                    6/6
Running            6              62d
openshift-monitoring                            prometheus-operator-admission-webhook-7d4759d465-mv98x   1/1
Running            1              62d
openshift-monitoring                            thanos-querier-6d77dcb87-c4pr6                      6/6
Running            6              62d
openshift-multus                                multus-additional-cni-plugins-jbrv2                 1/1
Running            8              69d
openshift-multus                                multus-x2ddp                                        1/1
Running            8              69d
openshift-multus                                network-metrics-daemon-tg528                        2/2
Running            16             69d
openshift-network-diagnostics                   network-check-target-mqr4t                          1/1
Running            8              69d
openshift-operator-lifecycle-manager            collect-profiles-28216020-66xqc                     0/1
Completed          0              6m8s
openshift-ovn-kubernetes                        ovnkube-node-d4g2s                                  5/5
Running            37             69d
```

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                                jcnr-0-dp-contrail-vrouter-nodes        1         1         1       1
1           <none>                                            16d
contrail                                jcnr-0-dp-contrail-vrouter-nodes-vrdpdk 1         1         1       1
1           <none>                                            16d
jcnr                                    syslog-ng                               1         1         1       1
1           <none>                                            16d
openshift-cluster-node-tuning-operator  tuned                                   5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-dns                           dns-default                             5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-dns                           node-resolver                           5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-image-registry                node-ca                                 5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-ingress-canary                ingress-canary                          2         2         2       2
2           kubernetes.io/os=linux                            69d
openshift-machine-api                   ironic-proxy                            3         3         3       3
3           node-role.kubernetes.io/master=                   69d
openshift-machine-config-operator       machine-config-daemon                   5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-machine-config-operator       machine-config-server                   3         3         3       3
3           node-role.kubernetes.io/master=                   69d
openshift-monitoring                    node-exporter                           5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-multus                        multus                                  5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-multus                        multus-additional-cni-plugins           5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-multus                        network-metrics-daemon                  5         5         5       5
5           kubernetes.io/os=linux                            69d
openshift-network-diagnostics           network-check-target                    5         5         5       5
```

```
5           beta.kubernetes.io/os=linux                                     69d
openshift-ovn-kubernetes                 ovnkube-master                            3         3         3        3
3           beta.kubernetes.io/os=linux,node-role.kubernetes.io/master=   69d
openshift-ovn-kubernetes                 ovnkube-node                    5         5         5       5             5
beta.kubernetes.io/os=linux                                     69d
```

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                   jcnr-0-crpd           1/1     16d
openshift-monitoring   alertmanager-main     2/2     69d
openshift-monitoring   prometheus-k8s        2/2     69d
```

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

   a. View the *access the cRPD CLI* 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 *access the vRouter CLI* 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 two fabric interfaces 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 port   packets:864 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
```

```
            RX packets:864  bytes:75536 errors:0
            TX packets:13609  bytes:1419892 errors:0
            Drops:0

vif0/1      PCI: 0000:17:00.0 (Speed 25000, Duplex 1) NH: 6 MTU: 9000
            Type:Physical HWaddr:40:a6:b7:a0:f0:6c IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:TcL3L2Vof QOS:0 Ref:9
            RX port   packets:243886 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:17:00.0  Status: UP  Driver: net_ice
            RX packets:243886  bytes:20529529 errors:0
            TX packets:243244  bytes:20010274 errors:0
            Drops:2675
            TX port   packets:243244 errors:0

vif0/2      PCI: 0000:17:00.1 (Speed 25000, Duplex 1) NH: 7 MTU: 9000
            Type:Physical HWaddr:40:a6:b7:a0:f0:6d IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:TcL3L2Vof QOS:0 Ref:8
            RX port   packets:129173 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:17:00.1  Status: UP  Driver: net_ice
            RX packets:129173  bytes:11623158 errors:0
            TX packets:129204  bytes:11624377 errors:0
            Drops:0
            TX port   packets:129204 errors:0

vif0/3      PMD: ens1f0 NH: 10 MTU: 9000
            Type:Host HWaddr:40:a6:b7:a0:f0:6c IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:65535 Flags:L3L2DProxyEr QOS:-1 Ref:11 TxXVif:1
            RX device packets:242329  bytes:19965464 errors:0
            RX queue  packets:242329 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:242329  bytes:19965464 errors:0
            TX packets:241163  bytes:20324343 errors:0
            Drops:0
            TX queue  packets:241163 errors:0
            TX device packets:241163  bytes:20324343 errors:0

vif0/4      PMD: ens1f1 NH: 15 MTU: 9000
            Type:Host HWaddr:40:a6:b7:a0:f0:6d IPaddr:0.0.0.0
```

```
DDP: OFF SwLB: ON
Vrf:0 Mcast Vrf:65535 Flags:L3L2DProxyEr QOS:-1 Ref:11 TxXVif:2
RX device packets:129204  bytes:11624377 errors:0
RX queue  packets:129204 errors:0
RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
RX packets:129204  bytes:11624377 errors:0
TX packets:129173  bytes:11623158 errors:0
Drops:0
TX queue  packets:129173 errors:0
TX device packets:129173  bytes:11623158 errors:0
```

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

System Requirements for OpenShift Deployment

IN THIS SECTION

Read this section to understand the system, resource, port, and licensing requirements for installing Juniper Cloud-Native Router on the Red Hat OpenShift Container Platform (OCP).

Minimum Host System Requirements for OCP

Table 8 on page 99 lists the host system requirements for installing Cloud-Native Router on OCP.

Table 8: Minimum Host System Requirements for OCP

Component	Value/Version	Notes
CPU	Intel x86	The tested CPU is Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz 64 core
Host OS	RHCOS 4.13	
Kernel Version	RedHat Enterprise Linux (RHEL): 4.18.X	The tested kernel version for RHEL is 4.18.0-372.40.1.el8_6.x86_64
NIC	• Intel E810 with Firmware 4.00 0x80014411 1.3236.0 • Intel E810-CQDA2 with Firmware 4.000x800144111.32 36.0 • Intel XL710 with Firmware 9.00 0x8000cead 1.3179.0 • Mellanox ConnectX-6 • Mellanox ConnectX-7	Support for Mellanox NICs is considered a Juniper Technology Preview ("Tech Preview" on page 452) feature. When using Mellanox NICs, ensure your interface names do not exceed 11 characters in length. When using Mellanox NICs, follow the interface naming procedure in "Interface Naming for Mellanox NICs" on page 109.
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
OCP Version	4.13	
OVN-Kubernetes CNI		

Table 8: Minimum Host System Requirements for OCP *(Continued)*

Component	Value/Version	Notes
Multus	Version 3.8	
Helm	3.12.x	
Container-RT	crio 1.25x	Other container runtimes may work but have not been tested with JCNR.
NOTE: The component versions listed in this table are expected to work with JCNR, but not every version or combination is tested in every release.		

Resource Requirements for OCP

Table 9 on page 100 lists the resource requirements for installing Cloud-Native Router on OCP.

Table 9: Resource Requirements for OCP

Resource	Value	Usage Notes
Data plane forwarding cores	1 core (1P + 1S)	
Service/Control Cores	0	

Table 9: Resource Requirements for OCP *(Continued)*

<table>
<tr><th>Resource</th><th>Value</th><th>Usage Notes</th></tr>
<tr><td>UIO Driver</td><td>VFIO-PCI</td><td>To enable, follow the steps below:
Create a Butane config file, <code>100-worker-vfiopci.bu</code>, binding the PCI device to the VFIO driver.
<pre>variant: openshift
version: 4.8.0
metadata:
  name: 100-worker-vfiopci
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
  - path: /etc/modprobe.d/vfio.conf
    mode: 0644
    overwrite: true
    contents:
      inline: |
        options vfio-pci ids=10de:1eb8
  - path: /etc/modules-load.d/vfio-pci.conf
    mode: 0644
    overwrite: true
    contents:
      inline: vfio-pci</pre>Create and apply the machine config:
<pre>$ butane 100-worker-vfiopci.bu -o 100-worker-vfiopci.yaml</pre><pre>$ oc apply -f 100-worker-vfiopci.yaml</pre></td></tr>
</table>

Table 9: Resource Requirements for OCP *(Continued)*

Resource	Value	Usage Notes

Hugepages (1G)	6 Gi	Configure huge pages on the worker nodes using the following commands:
		`oc create -f hugepages-tuned-boottime.yaml`
		
		`# cat hugepages-tuned-boottime.yaml`
		`apiVersion: tuned.openshift.io/v1`
		`kind: Tuned`
		`metadata:`
		`  name: hugepages`
		`  namespace: openshift-cluster-node-tuning-operator`
		`spec:`
		`  profile:`
		`  - data: |`
		`      [main]`
		`      summary=Boot time configuration for hugepages`
		`      include=openshift-node`
		`      [bootloader]`
		`      cmdline_openshift_node_hugepages=hugepagesz=1G hugepages=6`
		`    name: openshift-node-hugepages`
		`  recommend:`
		`  - machineConfigLabels:`
		`      machineconfiguration.openshift.io/role: "worker-hp"`
		`    priority: 30`
		`    profile: openshift-node-hugepages`
		
		`oc create -f hugepages-mcp.yaml`
		
		`# cat hugepages-mcp.yaml`
		`apiVersion: machineconfiguration.openshift.io/v1`
		`kind: MachineConfigPool`
		`metadata:`
		`  name: worker-hp`
		`  labels:`
		`    worker-hp: ""`
		`spec:`
		`  machineConfigSelector:`
		`    matchExpressions:`
		`      - {key: machineconfiguration.openshift.io/role, operator:`
		`In, values: [worker,worker-hp]}`
		`  nodeSelector:`

Table 9: Resource Requirements for OCP *(Continued)*

Resource	Value	Usage Notes
		`matchLabels:` `  node-role.kubernetes.io/worker-hp: ""`
Cloud-Native Router Controller cores	.5	
Cloud-Native Router vRouter Agent cores	.5	

Miscellaneous Requirements for OCP

Table 10 on page 103 lists additional requirements for installing Cloud-Native Router on OCP.

Table 10: Miscellaneous Requirements for OCP

	Cloud-Native Router Release Miscellaneous Requirements
Enable the host with SR-IOV and VT-d in the system's BIOS.	Depends on BIOS.

Enable VLAN driver at system boot.	Configure **/etc/modules-load.d/vlan.conf** as follows:
	`cat /etc/modules-load.d/vlan.conf`
	`8021q`
	Reboot and verify by executing the command:
	`lsmod | grep 8021q`

Table 10: Miscellaneous Requirements for OCP *(Continued)*

	Cloud-Native Router Release Miscellaneous Requirements

Enable VFIO-PCI driver at system boot.	Configure **/etc/modules-load.d/vfio.conf** as follows:
	`cat /etc/modules-load.d/vfio.conf`
	`vfio`
	`vfio-pci`
	Reboot and verify by executing the command:
	`lsmod | grep vfio`

Set IOMMU and IOMMU-PT.	Create a MachineConfig object that sets IOMMU and IOMMU-PT:
	`apiVersion: machineconfiguration.openshift.io/v1`
	`kind: MachineConfig`
	`metadata:`
	`  labels:`
	`    machineconfiguration.openshift.io/role: worker`
	`  name: 100-worker-iommu`
	`spec:`
	`  config:`
	`    ignition:`
	`      version: 3.2.0`
	`  kernelArguments:`
	`      - intel_iommu=on iommmu=pt`
	
	`$ oc create -f 100-worker-kernel-arg-iommu.yaml`

Disable spoofcheck on VFs allocated to JCNR. **NOTE**: Applicable for L2 deployments only.	`ip link set <interfacename> vf 1 spoofcheck off.`
Set trust on VFs allocated to JCNR. **NOTE**: Applicable for L2 deployments only.	`ip link set <interfacename> vf 1 trust on`

Table 10: Miscellaneous Requirements for OCP *(Continued)*

	Cloud-Native Router Release Miscellaneous Requirements

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.
 	
NOTE: Applicable for L3 deployments only.
	Create a conf file and add the kernel modules:
	
	`cat /etc/modules-load.d/crpd.conf`
	`tun`
	`fou`
	`fou6`
	`ipip`
	`ip_tunnel`
	`ip6_tunnel`
	`mpls_gso`
	`mpls_router`
	`mpls_iptunnel`
	`vrf`
	`vxlan`

Enable kernel-based forwarding on the Linux host.	`ip fou add port 6635 ipproto 137`

Table 10: Miscellaneous Requirements for OCP *(Continued)*

<table>
<tr><th></th><th>Cloud-Native Router Release Miscellaneous Requirements</th></tr>
<tr><td>Exclude Cloud-Native Router interfaces from NetworkManager control.</td><td>
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 Kubernetes management and create problems.
To avoid NetworkManager from interfering with Cloud-Native Router interface configuration, exclude Cloud-Native Router interfaces from NetworkManager control. Here's an example on how to do this in some Linux distributions:

Create the /etc/NetworkManager/conf.d/crpd.conf file and list the interfaces that you don't want NetworkManager to manage.
For example:
<pre><code>[keyfile]
 unmanaged-devices+=interface-name:enp*;interface-name:ens*</code></pre>
where <code>enp*</code> and <code>ens*</code> refer to your Cloud-Native Router interfaces.
NOTE: <code>enp*</code>
indicates all interfaces starting with
<code>enp</code>
. For specific interface names, provided a comma-separated list.
Restart the NetworkManager service:
<pre><code>sudo systemctl restart NetworkManager</code></pre>
.
Edit the /etc/sysctl.conf file on the host and paste the following content in it:
<pre><code>net.ipv6.conf.default.addr_gen_mode=0
net.ipv6.conf.all.addr_gen_mode=0</code></pre>

</td></tr>
</table>

Table 10: Miscellaneous Requirements for OCP *(Continued)*

	Cloud-Native Router Release Miscellaneous Requirements

	`net.ipv6.conf.default.autoconf=0`
	`net.ipv6.conf.all.autoconf=0`
	
	4. Run the command `sysctl -p /etc/sysctl.conf` to load the new **sysctl.conf** values on the host.
	5. 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`

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 11: 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 the Cloud-Native Router vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from Cloud-Native Router control plane
TCP	8075, 8076	Telemetry Information- Used for gNMI requests
TCP	9091	vRouter health check–cloud-native router checks to ensure the vRouter agent is running.
TCP	9092	vRouter health check–cloud-native router checks to ensure the vRouter DPDK is running.
TCP	50052	gRPC port–Cloud-Native Router listens on both IPv4 and IPv6
TCP	8081	Cloud-Native Router Deployer Port
TCP	24	cRPD SSH
TCP	830	cRPD NETCONF
TCP	666	**rpd**
TCP	1883	Mosquito mqtt–Publish/subscribe messaging utility
TCP	9500	**agentd** on cRPD

Table 11: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	21883	**na-mqttd**
TCP	50053	Default gNMI port that listens to the client subscription request
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Interface Naming for Mellanox NICs

When deploying Mellanox NICs in an OpenShift cluster, a conflict can arise between how OCP and Cloud-Native Router use interface names on those NICs. This might prevent your cluster from coming up.

Prior to installing JCNR, either disable predictable interface naming ("Option 1: Disable predictable interface naming" on page 109) or rename the Cloud-Native Router interfaces ("Option 2: Rename the Cloud-Native Router interfaces" on page 110). The Cloud-Native Router interfaces are the interfaces that you want Cloud-Native Router to control.

Option 1: Disable predictable interface naming

1. Before you start, ensure you have console access to the node.

2. Edit **/etc/default/grub** and append `net.ifnames=0` to `GRUB_CMDLINE_LINUX_DEFAULT`.

```
GRUB_CMDLINE_LINUX_DEFAULT="<existing_parameter_settings> net.ifnames=0"
```

3. Update grub.

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

4. Reboot the node.

5. Log back into the node. You might have to do this through the console if the network interfaces don't come back up.

6. List the interfaces and take note of the names of the non-Cloud-Native Router and Cloud-Native Router interfaces.

```
ip address
```

7. For all the non-Cloud-Native Router interfaces, update NetworkManager (or your network renderer) with the new interface names and restart NetworkManager.

8. Repeat on all the nodes where you're installing the Cloud-Native Router vRouter.

NOTE: Remember to update the fabric interfaces in your Cloud-Native Router installation helm chart with the new names of the Cloud-Native Router interfaces (or use subnets).

Option 2: Rename the Cloud-Native Router interfaces

1. Create a **/etc/udev/rules.d/00-persistent-net.rules** file to contain the rules.

2. Add the following line to the file:

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{address}=="<mac_address>",
ATTR{dev_id}=="0x0", ATTR{type}=="1", NAME="<new_ifname>"
```

   where *<mac_address>* is the MAC address of the interface you're renaming and *<new_ifname>* is the new name you want to assign to the interface (for example, `jcnr-eth1`).

3. Add a corresponding line for each interface you're renaming. (You're renaming all the interfaces that Cloud-Native Router controls.)

4. Reboot the node.

5. Repeat on all the nodes where you're installing the Cloud-Native Router vRouter.

NOTE: Remember to update the fabric interfaces in your Cloud-Native Router installation helm chart with the new names of the Cloud-Native Router interfaces (or use subnets).

Download Options

See "Cloud-Native Router Software Download Packages" on page 387.

Cloud-Native Router Licensing

See "Manage Cloud-Native Router Licenses" on page 352.

Customize Cloud-Native Router Helm Chart for OpenShift Deployment

IN THIS SECTION

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:

- 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_`*`release-number`*`/helmchart/values.yaml` file. The configuration keys of the helm chart are shown in the table below.

Helm Chart Description for OpenShift Deployment

Customize the Helm chart using the **Juniper_Cloud_Native_Router_*<release>*/helmchart/jcnr/values.yaml** file. We provide a copy of the default **values.yaml** in "Cloud-Native Router Default Helm Chart" on page 389.

Table 12 on page 112 contains a description of the configurable attributes in **values.yaml** for an OpenShift deployment.

Table 12: Helm Chart Description for OpenShift Deployment

Key	Description
global	
installSyslog	Set to true to install syslog-ng.
registry	Defines the Docker registry for the Cloud-Native Router container images. The default value is `enterprise-hub.juniper.net`. The images provided in the tarball are tagged with the default registry name. If you choose to host the container images to a private registry, replace the default value with your registry URL.
repository	(Optional) Defines the repository path for the Cloud-Native Router container images. This is a global key that takes precedence over the `repository` paths under the `common` section. Default is `jcnr-container-prod/`.
imagePullSecret	(Optional) Defines the Docker registry authentication credentials. You can configure credentials to either the Juniper Networks enterprise-hub.juniper.net registry or your private registry.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
registryCredentials	Base64 representation of your Docker registry credentials. See "Configure Repository Credentials" on page 398 for more information.
secretName	Name of the secret object that will be created.
common	Defines repository paths and tags for the various Cloud-Native Router container images. Use default unless using a private registry.
repository	Defines the repository path. The default value is `jcnr-container-prod/`. 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.
readinessCheck	Set to true to enable Cloud-Native Router Readiness preflight and postflight checks during installation. Comment this out or set to false to disable Cloud-Native Router Readiness preflight and postflight checks. Preflight checks verify that your infrastructure can support JCNR. Preflight checks take place before Cloud-Native Router is installed. Postflight checks verify that your Cloud-Native Router installation is working properly. Postflight checks take place after Cloud-Native Router is installed. See "Cloud-Native Router Readiness Checks" on page 362.
replicas	(Optional) Indicates the number of replicas for cRPD. Default is 1. The value for this key must be specified for multi-node clusters. The value is equal to the number of nodes running JCNR.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
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 to L2 and L2-L3 deployments. Enabling this key provides the functionality on all access interfaces. To enable the functionality on trunk interfaces, configure `no-local-switching` in `fabricInterface`. See *Prevent Local Switching* for more details.
iamRole	Not applicable.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>fabricInterface</td><td>
Aggregated interfaces that receive traffic from multiple interfaces. Fabric interfaces are always physical interfaces. They can either be a physical function (PF) or a virtual function (VF). The throughput requirement for these interfaces is higher — hence multiple hardware queues are allocated to them. Each hardware queue is allocated with a dedicated CPU core. See "Cloud-Native Router Interfaces Overview" on page 14 for more information.
Use this field to provide a list of fabric 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 the 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 <code>interface_mode</code> key configured, then the mode of deployment is L2.
When one or more interfaces have an <code>interface_mode</code> key configured along with the rest of the interfaces not having the <code>interface_mode</code> key, then the mode of deployment is L2-L3.
When none of the interfaces have the <code>interface_mode</code> key configured, then the mode of deployment is L3.

For example:
<pre>
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

L3 only
- eth1:
    ddp: "off"
</pre>
</td></tr>
</table>

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td></td><td><pre><code># 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</code></pre></td></tr>
<tr><td>subnet</td><td>An alternative mode of input to interface names. For example:<pre><code>- subnet: 10.40.1.0/24
  gateway: 10.40.1.1
  ddp: "off"</code></pre>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 in a multi-node cluster.</td></tr>
<tr><td>ddp</td><td>(Optional) Indicates the interface-level Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at the 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. See Enabling Dynamic Device Personalization (DDP) on Individual Interfaces for more details.

Options include auto, on, or off. Default is off.

NOTE: The interface level `ddp` takes precedence over the global `ddp` configuration.</td></tr>
</table>

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	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 the desired storm control profile to 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`
	
	See *Native VLAN* for more details.

no-local-switching	Prevents interfaces from communicating directly with each other when configured. Allowed values are true or false. See *Prevent Local Switching* for more details.
qosSchedulerProfileName	Specifies the QoS scheduler profile applicable to this interface. See the `qosSchedulerProfiles` section. If you don't specify a profile, then the QoS scheduler is disabled for this interface, which means that packets are scheduled with no regard to traffic class.

Table 12: Helm Chart Description for OpenShift Deployment ***(Continued)***

Key	Description
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 it set to the default INFO unless instructed to change it by Juniper Networks support.
log_path	The defined directory stores various JCNR-related descriptive logs such as **contrail-vrouter-agent.log**, **contrail-vrouter-dpdk.log**, etc. Default is **/var/log/jcnr/**.
syslog_notifications	Indicates the absolute path to the file that stores `syslog-ng` generated notifications in JSON format. Default is **/var/log/jcnr/jcnr_notifications.json**.
corePattern	Indicates the `core_pattern` for the core file. If left blank, then Cloud-Native Router pods will not overwrite the default pattern on the host. **NOTE**: Set the `core_pattern` on the host before deploying JCNR. You can 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 to the `core` file. Default is **/var/crash**.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description

nodeAffinity	(Optional) Defines labels on nodes to determine where to place the vRouter pods.
	
	By default the vRouter pods are deployed to all nodes of a cluster.
	
	In the example below, the node affinity label is defined as `key1=jcnr`. You must apply this label to each node where Cloud-Native Router is to be deployed:
	
	`nodeAffinity:`
	`- key: key1`
	`operator: In`
	`values:`
	`- jcnr`
	
	On an OCP setup, node affinity must be configured to bring up Cloud-Native Router on worker nodes only. For example:
	
	`nodeAffinity:`
	`- key: node-role.kubernetes.io/worker`
	`  operator: Exists`
	`- key: node-role.kubernetes.io/master`
	`  operator: DoesNotExist`
	
	NOTE: This key is a global setting.

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	For Red Hat OpenShift, don't leave this field empty. Set to **/var/lib/cni/bin**, which is the default path on any OCP deployment.
grpcTelemetryPort	(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50053.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
grpcVrouterPort	(Optional) Default is 50052. Configure to override.
vRouterDeployerPort	(Optional) Default is 8081. Configure to override.
jcnr-vrouter	
cpu_core_mask	If present, this indicates that you want to use static CPU allocation to allocate cores to the forwarding plane. This value should be a comma-delimited list of isolated CPU cores that you want to statically allocate to the forwarding plane (for example, `cpu_core_mask: "2,3,22,23"`). Use the cores not used by the host OS. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Cloud-Native Router Readiness preflight checks, if enabled, will fail the installation if you specify both.
guaranteedVrouterCpus	If present, this indicates that you want to use the Kubernetes CPU Manager to allocate CPU cores to the forwarding plane. This value should be the number of guaranteed CPU cores that you want the Kubernetes CPU Manager to allocate to the forwarding plane. You should set this value to at least one more than the number of forwarding cores. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. The installation will fail if you specify both.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
dpdkCtrlThreadMask	Specifies the CPU core(s) to allocate to vRouter DPDK control threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `serviceCoreMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.
serviceCoreMask	Specifies the CPU core(s) to allocate to vRouter DPDK service threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `dpdkCtrlThreadMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.
numServiceCtrlThreadCPU	Specifies the number of CPU cores to allocate to vRouter DPDK service/control traffic when using the Kubernetes CPU Manager. This number should be smaller than the number of `guaranteedVrouterCpus` cores. The remaining `guaranteedVrouterCpus` cores are allocated for forwarding. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
numberOfSchedulerLcores	The number of CPU cores that you want Kubernetes CPU Manager to dedicate to your QoS schedulers. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
restoreInterfaces	Set to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts or if Cloud-Native Router is uninstalled.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
bondInterfaceConfigs	(Optional) Enable bond interface configurations only for L2 or L2-L3 deployments.
name	Name of the bond interface.
mode	Set to 1 (active-backup).
slaveInterfaces	List of fabric interfaces to be bonded.
primaryInterface	(Optional) Primary interface for the bond.
slaveNetworkDetails	Not applicable.
mtu	Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default is 9000.
qosSchedulerProfiles	Defines the QoS scheduler profiles that are referenced from the `fabricInterface` section.
sched_profile_1	The name of the QoS scheduler profile.
	cpu Specify the CPU core(s) to dedicate to the scheduler. If `cpu_core_mask` is specified, this should be a unique additional core(s). **bandwidth** Specify the bandwidth in Gbps.
stormControlProfiles	Configure the rate limit profiles for BUM traffic on fabric interfaces in bytes per second. See */Content/l2-bum-rate-limiting_xi931744_1_1.dita* for more details.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
dpdkCommandAdditionalArgs	Pass any additional DPDK command line parameters. The `--yield_option 0` is set by default and implies the DPDK forwarding cores will not yield their assigned CPU cores. Other common parameters that can be added are `tx` and `rx` descriptors and `mempool`. For example: `dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 2048 --dpdk_rxd_sz 2048 --vr_mempool_sz 131072"` **NOTE**: Changing the number of `tx` and `rx` descriptors and the `mempool` size affects the number of huge pages required. If you make explicit changes to these parameters, set the number of huge pages to 10 (x 1 GB). See Table 9 on page 100 in "System Requirements for OpenShift Deployment" on page 98 for information on how to configure huge pages on an OpenShift node and see "Configure the Number of Huge Pages to Use" on page 403 for information on how to configure the number of huge pages that the Cloud-Native Router vRouter uses.
dpdk_monitoring_thread_config	(Optional) Enables a monitoring thread for the vRouter DPDK container. Every `loggingInterval` seconds, a log containing the information indicated by `loggingMask` is generated.
loggingMask	Specifies the information to be generated. Represented by a bitmask with bit positions as follows: • 0b001 is the nl_counter • 0b010 is the lcore_timestamp • 0b100 is the profile_histogram
loggingInterval	Specifies the log generation interval in seconds.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
ddp	(Optional) Indicates the global Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at the 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. See *Enabling Dynamic Device Personalization (DDP) on Individual Interfaces* for more details. Options include auto, on, or off. Default is off. **NOTE**: The interface level `ddp` takes precedence over the global `ddp` configuration.
twampPort	(Optional) The TWAMP session reflector port (if you want TWAMP sessions to use vRouter timestamps). The vRouter listens to TWAMP test messages on this port and inserts/overwrites timestamps in TWAMP test messages. Timestamping TWAMP messages at the vRouter (instead of at cRPD) leads to more accurate measurements. Valid values are 862 and 49152 through 65535. If this parameter is absent, then the vRouter does not insert or overwrite timestamps in the TWAMP session. Timestamps are taken and inserted by cRPD instead. See *Two-Way Active Measurement Protocol (TWAMP)*.
vrouter_dpdk_uio_driver	The uio driver is `vfio-pci`.
agentModeType	Set to dpdk.
fabricRpfCheckDisable	Set to false to enable the RPF check on all Cloud-Native Router fabric interfaces. By default, RPF check is disabled.
telemetry	(Optional) Configures cRPD telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
disable	Set to true to disable cRPD telemetry. Default is false, which means that cRPD telemetry is enabled by default.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
metricsPort	The port that the cRPD telemetry exporter is listening to Prometheus queries on. Default is 8072.
logLevel	One of warn, warning, info, debug, trace, or verbose. Default is info.
gnmi	(Optional) Configures cRPD gNMI settings.
	enable Set to true to enable the cRPD telemetry exporter to respond to gNMI requests.
vrouter	
telemetry	(Optional) Configures vRouter telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
	metricsPort Specify the port that the vRouter telemetry exporter listens to Prometheus queries on. Default is 8070.
	logLevel One of warn, warning, info, debug, trace, or verbose. Default is info.
	gnmi (Optional) Configures vRouter gNMI settings. enable - Set to true to enable the vRouter telemetry exporter to respond to gNMI requests.
persistConfig	Set to true if you want Cloud-Native Router pod configuration to persist even after uninstallation. This option can only be set for L2 mode deployments. Default is false.
enableLocalPersistence	Set to true if you're using the cRPD CLI or NETCONF to configure JCNR. When set to true, the cRPD CLI and NETCONF configuration persists through node reboots, cRPD pod restarts, and Cloud-Native Router upgrades. Default is false.

Table 12: Helm Chart Description for OpenShift Deployment *(Continued)*

Key	Description
interfaceBoundType	Not applicable.
networkDetails	Not applicable.
networkResources	Not applicable.
contrail-tools	
install	Set to true to install contrail-tools (used for debugging).

Customize Cloud-Native Router Configuration

SUMMARY

Read this topic to understand how to customize Cloud-Native Router configuration using a Configlet custom resource.

IN THIS SECTION

Configlet Custom Resource

Starting with Juniper Cloud-Native Router (JCNR) Release 24.2, we support customizing Cloud-Native Router configuration using a configlet custom resource. The configlet can be generated either by rendering a predefined template of supported Junos configuration or using raw configuration. The generated configuration is validated and deployed on the Cloud-Native Router controller (cRPD) as one or more Junos configuration groups.

NOTE: You can configure Cloud-Native Router using either configlets or the cRPD CLI or NETCONF. If you use the cRPD CLI or NETCONF, be sure to enable local persistence in **values.yaml** (`enableLocalPersistence: true`) so that your CLI or NETCONF configuration persists across reboots and upgrades.

NOTE: Using both configlets and the cRPD CLI or NETCONF to configure Cloud-Native Router may lead to unpredictable behavior. Use one or the other, but not both.

Configuration Examples

You create a configlet custom resource of the kind `Configlet` in the `jcnr` namespace. You provide raw configuration as Junos `set` commands.

Use `crpdSelector` to control where the configlet applies. The generated configuration is deployed to cRPD pods on nodes matching the specified label only. If `crpdSelector` is not defined, the configuration is applied to all cRPD pods in the cluster.

An example configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample             # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker                    # <-- Node label to select the cRPD pods
```

You can also use a templatized configlet yaml that contains keys or variables. The values for variables are provided by a configletDataValue custom resource, referenced by `configletDataValueRef` . An example templatized configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
```

```
metadata:
  name: configlet-sample-with-template      # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: worker                          # <-- Node label to select the cRPD pods
  configletDataValueRef:
    name: "configletdatavalue-sample"       # <-- Configlet Data Value resource name
```

To render configuration using the template, you must provide key:value pairs in the ConfigletDataValue custom resource:

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"           # <-- Key:Value pair
  }
```

The generated configuration is validated and applied to all or selected cRPD pods as a Junos Configuration Group.

Applying the Configlet Resource

The configlet resource can be used to apply configuration to selected or all cRPD pods either when Cloud-Native Router is deployed or once the cRPD pods are up and running. Let us look at configlet deployment in detail.

Applying raw configuration

1. Create raw configuration configlet yaml. The example below configures a loopback interface in cRPD.

```
cat configlet-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker
```

2. Apply the configuration using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                              AGE
configlet-sample-node1            10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

For example:

```
kubectl describe nodeconfiglet configlet-sample-node1 -n jcnr
```

The following output has been trimmed for brevity:

```
Name:         configlet-sample-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
Annotations:  <none>
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 10.10.10.1/32;
            }
        }
```

```
    }
}
```

NOTE: The configuration generated using configlets is applied to cRPD as configuration groups. We therefore recommend that you not use configuration groups when specifying your configlet.

Applying templatized configuration

1. Create the templatized configlet yaml and the configlet data value yaml for key:value pairs.

```
cat configlet-sample-template.yaml
```


```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-template
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: master
  configletDataValueRef:
    name: "configletdatavalue-sample"
```


```
cat configletdatavalue-sample.yaml
```


```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
```

```
    "Ip": "127.0.0.1"
  }
```

2. Apply the configuration using the `kubectl apply` command, starting with the config data value yaml.

```
kubectl apply -f configletdatavalue-sample.yaml
```

```
configletdatavalue.configplane.juniper.net/configletdatavalue-sample created
```

```
kubectl apply -f configlet-sample-template.yaml
```

```
configlet.configplane.juniper.net/configlet-sample-template created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                                      AGE
configlet-sample-template-node1           10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

```
kubectl describe nodeconfiglet configlet-sample-template-node1 -n jcnr
```

   The following output has been trimmed for brevity:

```
Name:         configlet-sample-template-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
```

```
Annotations:  <none>
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample-template
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample-template
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32;
            }
        }
    }
}
```

Modifying the Configlet

You can modify a configlet resource by changing the yaml file and reapplying it using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample configured
```

Any changes to existing configlet resource are reconciled by replacing the configuration group on cRPD.

You can delete the configuration group by deleting the configlet resource using the `kubectl delete` command.

```
kubectl delete configlet configlet-sample -n jcnr
```

```
configlet.configplane.juniper.net "configlet-sample" deleted
```

Troubleshooting

If you run into problems, check the `contrail-k8s-deployer` logs. For example:

```
kubectl logs contrail-k8s-deployer-8ff895cc5-cbfwm -n contrail-deploy
```

4

CHAPTER

Install Cloud-Native Router on Amazon EKS

IN THIS CHAPTER

Install and Verify Juniper Cloud-Native Router on Amazon EKS

IN THIS SECTION

The Juniper 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.

Install Juniper Cloud-Native Router Using Juniper Support Site Package

Read this section to learn the steps required to install the cloud-native router components using Helm charts.

1. Review the "System Requirements for EKS Deployment" on page 149 to ensure the setup has all the required configuration.
2. Download the desired Cloud-Native Router software package to the directory of your choice.

   You have the option of downloading the package to install Cloud-Native Router only or downloading the package to install JNCR together with Juniper cSRX. See "Cloud-Native Router Software Download Packages" on page 387 for a description of the packages available. If you don't want to install Juniper cSRX now, you can always choose to install Juniper cSRX on your working Cloud-Native Router installation later.
3. Expand the file **Juniper_Cloud_Native_Router_*<release-number>*.tgz**.

   ```
   tar xzvf Juniper_Cloud_Native_Router_<release-number>.tgz
   ```
4. Change directory to the main installation directory.

- If you're installing Cloud-Native Router only, then:

  ```
  cd Juniper_Cloud_Native_Router_<release>
  ```

  This directory contains the Helm chart for Cloud-Native Router only.

- If you're installing Cloud-Native Router and cSRX at the same time, then:

  ```
  cd Juniper_Cloud_Native_Router_CSRX_<release>
  ```

  This directory contains the combination Helm chart for Cloud-Native Router and cSRX.

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

5. View the contents in the current directory.

   ```
   ls
   helmcharts  images  README.md  secrets
   ```

6. Change to the **helmchart** directory and expand the Helm chart.

   ```
   cd helmchart
   ```

   - For Cloud-Native Router only:

     ```
     ls
     jcnr-<release>.tgz
     ```

     ```
     tar -xzvf jcnr-<release>.tgz
     ```

     ```
     ls
     jcnr   jcnr-<release>.tgz
     ```

     The Helm chart is located in the **jcnr** directory.

- For the combined Cloud-Native Router and cSRX:

```
ls
jcnr_csrx-<release>.tgz
```

```
tar -xzvf jcnr_csrx-<release>.tgz
```

```
ls
jcnr_csrx   jcnr_csrx-<release>.tgz
```

The Helm chart is located in the **jcnr_csrx** directory.

7. Follow the steps in "Installing Your License" on page 353 to install your Cloud-Native Router license.

8. 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**.

9. Apply **secrets/jcnr-secrets.yaml** to the cluster.

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

10. Create the "JCNR ConfigMap" on page 153 if using the Virtual Router Redundancy Protocol (VRRP) for your Cloud-Native Router cluster. A sample `jcnr-aws-config.yaml` manifest is provided in

cRPD_examples directory in the installation bundle. Apply the jcnr-aws-config.yaml to the Kubernetes system.

```
kubectl apply -f jcnr-aws-config.yaml
configmap/jcnr-aws-config created
```

11. If desired, configure how cores are assigned to the vRouter DPDK containers. See "Allocate CPUs to the Cloud-Native Router Forwarding Plane" on page 355.
12. Customize the Helm chart for your deployment using the **helmchart/jcnr/values.yaml** or **helmchart/jcnr_csrx/values.yaml** file.

    See "Customize JCNR Helm Chart for EKS Deployment" on page 157 for descriptions of the helm chart configurations and a sample helm chart for EKS deployment.
13. Optionally, customize Cloud-Native Router configuration.

    See, "Customize Cloud-Native Router Configuration " on page 62 for creating and applying the cRPD customizations.
14. If you're installing Juniper cSRX now, then follow the procedure in "Apply the cSRX License and Configure cSRX" on page 338.
15. Install Multus CNI using the following command:

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/config/
multus/v3.7.2-eksbuild.1/aws-k8s-multus.yaml
```

16. Install the Amazon Elastic Block Storage (EBS) Container Storage Interface (CSI) driver.
17. Label the nodes where you want Cloud-Native Router to be installed based on the nodeaffinity configuration (if defined in the values.yaml). For example:

```
kubectl label nodes ip-10.0.100.17.us-east-2.compute.internal key1=jcnr --overwrite
```

18. Deploy the Juniper Cloud-Native Router using the Helm chart.

    Navigate to the **helmchart/jcnr** or the **helmchart/jcnr_csrx** directory and run the following command:

```
helm install jcnr .
```

or

```
helm install jcnr-csrx .
```

```
NAME: jcnr
LAST DEPLOYED: Fri Dec 22 06:04:33 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

19. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

Sample output:

```
NAME        NAMESPACE    REVISION        UPDATED
STATUS          CHART                      APP VERSION
jcnr        default      1               2023-12-22 06:04:33.144611017 -0400 EDT
deployed        jcnr-<version>   <version>
```

Install Juniper Cloud-Native Router Using AWS Marketplace Subscription (BYOL)

Use this procedure to install JCNR (BYOL) from AWS Marketplace using Helm charts.

This procedure installs Cloud-Native Router on your existing Amazon EKS cluster. Ensure you've set up your Amazon EKS cluster prior to running this procedure. You can use any method to create an EKS cluster as long as it meets the system requirements described in "System Requirements for EKS Deployment" on page 149.

For convenience, we've provided a CloudFormation template that you can use to quickly get a cluster up and running. This template is provided in "CloudFormation Template for EKS Cluster" on page 406.

1. Review the "System Requirements for EKS Deployment" on page 149 to ensure the setup has all the required configuration.
2. Log in to and search for Cloud-Native Router products from the AWS Marketplace.
3. Select the JCNR (BYOL) product and subscribe to it.
4. Scroll down on the selected product's landing page to view the usage instructions.

   The instructions show you how to log in to the ECR Helm registry and download the Cloud-Native Router helm chart.
5. Copy and run the provided usage instructions on the setup where you issue your AWS CLI commands.

```
aws configure
aws ecr get-login-password <...>
helm pull oci: <...>
```

   This downloads the **jcnr-*<version>*.tgz** file onto your setup.
6. Expand the file **jcnr-*<version>*.tgz**.

```
tar xzvf jcnr-<version>.tgz
```

7. Change directory to `jcnr`.

```
cd jcnr
```

NOTE: All remaining steps in the installation assume that your current working directory is now **jcnr**.

8. View the contents in the current directory.

```
ls
Chart.yaml  charts  cRPD_examples  scripts  secrets  values.yaml
```

9. Follow the steps in "Installing Your License" on page 353 to install your Cloud-Native Router license.

10. 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**.

11. Apply **secrets/jcnr-secrets.yaml** to the cluster.

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

12. Create the "JCNR ConfigMap" on page 153 if using the Virtual Router Redundancy Protocol (VRRP) for your Cloud-Native Router cluster. Apply the `jcnr-aws-config.yaml` to the Kubernetes system.

```
kubectl apply -f jcnr-aws-config.yaml
configmap/jcnr-aws-config created
```

13. If desired, configure how cores are assigned to the vRouter DPDK containers. See "Allocate CPUs to the Cloud-Native Router Forwarding Plane" on page 355.

14. Customize the helm chart for your deployment using the **values.yaml** file.

    See, "Customize JCNR Helm Chart for EKS Deployment" on page 157 for descriptions of the helm chart configurations and a sample helm chart for EKS deployment.

15. Optionally, customize Cloud-Native Router configuration.

    See "Customize Cloud-Native Router Configuration " on page 62 for creating and applying the cRPD customizations.

16. Verify that the Amazon EBS CSI driver role policy has been attached to the EKS cluster node role.

```
aws iam list-attached-role-policies --role-name <EKS_Cluster_Node_Role_Name>
```

Look for `arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy` in the output. If this policy is not listed, add it as follows:

```
aws iam attach-role-policy --role-name <EKS_Cluster_Node_Role_Name> --policy-arn
arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
```

17. Verify that the Amazon VPC CNI role policy has been attached to the EKS cluster node role.

```
aws iam list-attached-role-policies --role-name <EKS_Cluster_Node_Role-Name>
```

Look for `arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy` in the output. If this policy is not listed, add it as follows:

```
aws iam attach-role-policy --role-name <EKS_Cluster_Node_Role_Name> --policy-arn
arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

18. Verify that the Amazon EBS CSI driver and Amazon VPC CNI add-ons are installed.

```
aws eks describe-addon-versions --addon-name aws-ebs-csi-driver
```

```
aws eks describe-addon-versions --addon-name vpc-cni
```

If any of the add-ons is not installed, you can install them respectively as follows:

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-ebs-csi-driver --addon-
version <version> --service-account-role-arn <EKS_Cluster_Node_IAM_role_ARN>
```

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
<version> --service-account-role-arn <EKS_Cluster_Node_IAM_role_ARN>
```

Be sure to install the versions listed in "Minimum Host System Requirements for EKS" on page 149.

19. Label the nodes where you want Cloud-Native Router to be installed based on the `nodeaffinity` configuration (if defined in the `values.yaml`). For example:

```
kubectl label nodes ip-10.0.100.17.us-east-2.compute.internal key1=jcnr --overwrite
```

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

    Run the following command:

```
helm install jcnr .
```

```
NAME: jcnr
LAST DEPLOYED: <date_time>
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

21. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

    Sample output:

```
NAME    NAMESPACE    REVISION    UPDATED        STATUS      CHART             APP VERSION
jcnr    default      1           <date_time>    deployed    jcnr-<version>    <version>
```

Verify Cloud-Native Router Installation on Amazon EKS

NOTE: The output shown in this example procedure is affected by the number of nodes in the cluster. The output you see in your setup may differ in that regard.

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-5b6c9656d5-nw9t9           1/1     Running   0               13d
contrail          jcnr-0-dp-contrail-vrouter-nodes-b2jxp           2/2     Running   0               13d
contrail          jcnr-0-dp-contrail-vrouter-nodes-vrdpdk-g7wrk    1/1     Running   0               13d
jcnr              jcnr-0-crpd-0                                    1/1     Running   0               13d
jcnr              syslog-ng-tct27                                  1/1     Running   0               13d
kube-system       aws-node-k8hxl                                   1/1     Running   1 (15d ago)     15d
kube-system       ebs-csi-node-c8rbh                               3/3     Running   3 (15d ago)     15d
kube-system       kube-multus-ds-8nzhs                             1/1     Running   1 (13d ago)     13d
kube-system       kube-proxy-h669c                                 1/1     Running   1 (15d ago)     15d
```

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      jcnr-0-dp-contrail-vrouter-nodes            1         1         1       1            1
<none>                        13d
contrail      jcnr-0-dp-contrail-vrouter-nodes-vrdpdk     1         1         1       1            1
<none>                        13d
jcnr          syslog-ng                                   1         1         1       1            1
<none>                        13d
kube-system   aws-node                                    8         8         8       8            8
<none>                        15d
kube-system   ebs-csi-node                                8         8         8       8            8           kubernetes.io/
os=linux      15d
kube-system   ebs-csi-node-windows                        0         0         0       0            0           kubernetes.io/
os=windows    15d
kube-system   kube-multus-ds                              8         8         8       8            8
```

```
<none>                  13d
kube-system   kube-proxy                              8         8         8       8           8
<none>                  15d
```

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        jcnr-0-crpd   1/1     27m
```

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

   a. View the *Access the cRPD CLI* section for instructions 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 *Access the vRouter CLI* section for instructions on how 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
            RX packets:0  bytes:0 errors:0
            TX packets:0  bytes:0 errors:0
            Drops:0

vif0/1      PCI: 0000:00:07.0 (Speed 1000, Duplex 1) NH: 6 MTU: 9000
```

```
            Type:Physical HWaddr:0e:d0:2a:58:46:4f IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:L3L2 QOS:0 Ref:8
            RX device packets:20476  bytes:859992 errors:0
            RX port   packets:20476 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:00:07.0  Status: UP  Driver: net_ena
            RX packets:20476  bytes:859992 errors:0
            TX packets:2  bytes:180 errors:0
            Drops:0
            TX port   packets:2 errors:0
            TX device packets:8  bytes:740 errors:0

vif0/2      PCI: 0000:00:08.0 (Speed 1000, Duplex 1) NH: 7 MTU: 9000
            Type:Physical HWaddr:0e:6a:9e:04:da:6f IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:0 Flags:L3L2 QOS:0 Ref:8
            RX device packets:20476  bytes:859992 errors:0
            RX port   packets:20476 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: 0000:00:08.0  Status: UP  Driver: net_ena
            RX packets:20476  bytes:859992 errors:0
            TX packets:2  bytes:180 errors:0
            Drops:0
            TX port   packets:2 errors:0
            TX device packets:8  bytes:740 errors:0

vif0/3      PMD: eth2 NH: 10 MTU: 9000
            Type:Host HWaddr:0e:d0:2a:58:46:4f IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
            Vrf:0 Mcast Vrf:65535 Flags:L3L2DProxyEr QOS:-1 Ref:11 TxXVif:1
            RX device packets:2  bytes:180 errors:0
            RX queue  packets:2 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
            RX packets:2  bytes:180 errors:0
            TX packets:20476  bytes:859992 errors:0
            Drops:0
            TX queue  packets:20476 errors:0
            TX device packets:20476  bytes:859992 errors:0

vif0/4      PMD: eth3 NH: 15 MTU: 9000
            Type:Host HWaddr:0e:6a:9e:04:da:6f IPaddr:0.0.0.0
            DDP: OFF SwLB: ON
```

```
Vrf:0 Mcast Vrf:65535 Flags:L3L2DProxyEr QOS:-1 Ref:11 TxXVif:2
RX device packets:2  bytes:180 errors:0
RX queue  packets:2 errors:0
RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0
RX packets:2  bytes:180 errors:0
TX packets:20476  bytes:859992 errors:0
Drops:0
TX queue  packets:20476 errors:0
TX device packets:20476  bytes:859992 errors:0
```

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

System Requirements for EKS Deployment

IN THIS SECTION

Read this section to understand the system, resource, port, and licensing requirements for installing Juniper Cloud-Native Router on Amazon Elastic Kubernetes Service (EKS).

Minimum Host System Requirements for EKS

Table 13 on page 150 lists the host system requirements for installing Cloud-Native Router on EKS.

Table 13: Minimum Host System Requirements for EKS

Component	Value/Version
EKS Deployment	Self-managed nodes or managed node group
Host OS	Amazon Linux 2
EKS version / Kubernetes	1.26.3, 1.28.x, 1.29.x
EC2 Instance Type	Any instance type with ENA adapters **NOTE**: There is no minimum instance type imposed by Juniper Cloud-Native Router, but a typical deployment runs c5.4xlarge or m5.4xlarge or larger (depending on performance requirements).
Kernel Version	5.10.x, 5.15.x
NIC	Elastic Network Adapter (ENA)
AWS CLI version	2.11.9
VPC CNI	v1.14.0-eksbuild.3
EBS CSI Driver	v1.28.0-eksbuild.1
Node Role	AmazonEBSCSIDriverPolicy AmazonEKS_CNI_Policy
Multus	3.7.2 (`kubectl apply -f https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/master/config/multus/v3.7.2-eksbuild.1/aws-k8s-multus.yaml`)
Helm	3.11
Container-RT	containerd 1.7.x

Table 13: Minimum Host System Requirements for EKS *(Continued)*

Component	Value/Version
NOTE: The component versions listed in this table are expected to work with JCNR, but not every version or combination is tested in every release.	

Resource Requirements for EKS

Table 14 on page 151 lists the resource requirements for installing Cloud-Native Router on EKS.

Table 14: Resource Requirements for EKS

Resource	Value	Usage Notes
Data plane forwarding cores	1 core (1P + 1S)	
Service/Control Cores	0	

UIO Driver	VFIO-PCI	To enable, follow the steps below:
		
		`cat /etc/modules-load.d/vfio.conf`
		`vfio`
		`vfio-pci`
		
		Enable Unsafe IOMMU mode
		
		`echo Y > /sys/module/vfio_iommu_type1/parameters/`
		`allow_unsafe_interrupts`
		`echo Y > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode`

Hugepages (1G)	6 Gi	See "Configure the Number of Huge Pages Available on a Node" on page 401.
Cloud-Native Router Controller cores	.5	

Table 14: Resource Requirements for EKS *(Continued)*

Resource	Value	Usage Notes
Cloud-Native Router vRouter Agent cores	.5	

Miscellaneous Requirements for EKS

Table 15 on page 152 lists additional requirements for installing Cloud-Native Router on EKS.

Table 15: Miscellaneous Requirements for EKS

Requirement	Example
Disable source/destination checks.	Disable source/destination checks on the AWS Elastic Network Interfaces (ENI) interfaces attached to JCNR. JCNR, being a transit router, is neither the source nor the destination of any traffic that it receives.
Attach IAM policy.	Attach the `AmazonEBSCSIDriverPolicy` IAM policy to the role assigned to the EKS cluster.

Set IOMMU and IOMMU-PT in GRUB.	Add the following line to **/etc/default/grub**.
	`GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
	Update grub and reboot.
	`grub2-mkconfig -o /boot/grub2/grub.cfg`
	`reboot`

Table 15: Miscellaneous Requirements for EKS *(Continued)*

Requirement	Example

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.
 	
NOTE: Applicable for L3 deployments only.
	Create a **/etc/modules-load.d/crpd.conf** file and add the following kernel modules to it:
	
	`tun`
	`fou`
	`fou6`
	`ipip`
	`ip_tunnel`
	`ip6_tunnel`
	`mpls_gso`
	`mpls_router`
	`mpls_iptunnel`
	`vrf`
	`vxlan`

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`

Cloud-Native Router ConfigMap for VRRP

You can enable Virtual Router Redundancy Protocol (VRRP) for your Cloud-Native Router cluster.

> **NOTE**: When running VRRP, the AWS IAM role for the node hosting the Cloud-Native Router instance needs permission to modify the VPC route table. To provide that permission, add the **NetworkAdministrator** policy to that IAM role.

You must create a Cloud-Native Router ConfigMap to define the behavior of VRRP for your Cloud-Native Router cluster in an EKS deployment. Considering that AWS VPC supports exactly one next-hop for a prefix, the ConfigMap defines how the VRRP mastership status is used to copy prefixes from routing tables in Cloud-Native Router to specific routing tables in AWS.

We provide an example `jcnr-aws-config.yaml` manifest below:

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: jcnr-aws-config
    namespace: jcnr
data:
  aws-rttable-map.json: |
    [
      {
        "jcnr-table-name":"default-rt.inet.0",
        "jcnr-policy-name": "default-rt-to-aws-export",
        "jcnr-nexthop-interface-name":"eth4",
        "vpc-table-tag":"jcnr-aws-vpc-internal-table"
      },
      {
        "jcnr-table-name":"default-rt.inet6.0",
        "jcnr-policy-name":"default-rt-to-aws-export",
        "jcnr-nexthop-interface-name":"eth4",
        "vpc-table-tag":"jcnr-aws-vpc-internal-table"
      }
    ]
```

Table 16 on page 154 describes the ConfigMap elements:

Table 16: Cloud-Native Router ConfigMap Elements

Element	Description
`jcnr-table-name`	The routing table in Cloud-Native Router from which prefixes should be copied.
`jcnr-policy-name`	A routing policy in Cloud-Native Router that imports the prefixes in the named routing table to copy to the AWS routing table.

Table 16: Cloud-Native Router ConfigMap Elements *(Continued)*

Element	Description
`jcnr-nexthop-interface-name`	Name of the Cloud-Native Router interface which should be used as the next-hop by the AWS VPC route table when this instance of the Cloud-Native Router is VRRP master.
`vpc-table-tag`	A freeform tag applied to the VPC route table in AWS to which the prefixes should be copied.

Apply `jcnr-aws-config.yaml` to the cluster before installing JCNR. The Cloud-Native Router CNI deployer renders the cRPD configuration based on the ConfigMap.

NOTE: When not using VRRP, provide an empty list as the data for `aws-rttable-map.json`.

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 17: 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 the Cloud-Native Router vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from Cloud-Native Router control plane
TCP	8075, 8076	Telemetry Information- Used for gNMI requests

Table 17: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	9091	vRouter health check–cloud-native router checks to ensure the vRouter agent is running.
TCP	9092	vRouter health check–cloud-native router checks to ensure the vRouter DPDK is running.
TCP	50052	gRPC port–Cloud-Native Router listens on both IPv4 and IPv6
TCP	8081	Cloud-Native Router Deployer Port
TCP	24	cRPD SSH
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	50053	Default gNMI port that listens to the client subscription request
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

To deploy Cloud-Native Router on an EKS cluster, you can either download the Helm charts from the Juniper Networks software download site (see "Cloud-Native Router Software Download Packages" on page 387) or subscribe via the AWS Marketplace.

> **NOTE**: Before deploying Cloud-Native Router on an EKS cluster via Helm charts downloaded from the Juniper Networks software download site, you must whitelist the https://enterprise.hub.juniper.net URL as the Cloud-Native Router image repository.

Cloud-Native Router Licensing

You can purchase BYOL licenses for the Juniper Cloud-Native Router software through your Juniper Account Team.

For information on BYOL licenses, see "Manage Cloud-Native Router Licenses" on page 352.

Customize Cloud-Native Router Helm Chart for EKS Deployment

IN THIS SECTION

Read this topic to learn about the deployment configuration available for the Juniper Cloud-Native Router when deployed on Amazon EKS.

You can deploy and operate Juniper Cloud-Native Router in the L3 mode on Amazon EKS. You configure the deployment mode by editing the appropriate attributes in the **values.yaml** file prior to deployment.

Helm Chart Description for Amazon EKS Deployment

Customize the Helm chart using the **Juniper_Cloud_Native_Router_*<release>*/helmchart/jcnr/values.yaml** file. We provide a copy of the default **values.yaml** in "Cloud-Native Router Default Helm Chart" on page 389.

Table 18 on page 158 contains a description of the configurable attributes in **values.yaml** for an Amazon EKS deployment.

Table 18: Helm Chart Description for Amazon EKS Deployment

Key	Description
global	
installSyslog	Set to true to install syslog-ng.
registry	Defines the Docker registry for the Cloud-Native Router container images. The default value is set to: • `enterprise-hub.juniper.net` in Helm charts downloaded from the Juniper Networks software download site. • Amazon Elastic Container Registry (ECR) for Helm charts downloaded from the AWS Marketplace.
repository	(Optional) Defines the repository path for the Cloud-Native Router container images. This is a global key that takes precedence over the `repository` paths under the `common` section. Default is `jcnr-container-prod/`.
imagePullSecret	(Optional) Defines the Docker registry authentication credentials. You can configure credentials to either the Juniper Networks enterprise-hub.juniper.net registry or your private registry. Not applicable for AWS Marketplace subscriptions.

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

Key	Description
registryCredentials	Base64 representation of your Docker registry credentials. See "Configure Repository Credentials" on page 398 for more information.
secretName	Name of the secret object that will be created.
common	Defines repository paths and tags for the vRouter, cRPD and jcnr-cni container images. Use the defaults.
repository	Defines the repository path. The global repository key takes precedence if defined. The default value is set to: • `jcnr-container-prod/` in Helm charts downloaded from the Juniper Networks software download site. • `juniper-networks` for AWS Marketplace subscriptions.
tag	Defines the image tag. The default value is configured to the appropriate tag number for the Cloud-Native Router release version.
readinessCheck	Set to true to enable Cloud-Native Router Readiness preflight and postflight checks during installation. Comment this out or set to false to disable Cloud-Native Router Readiness preflight and postflight checks. Preflight checks verify that your infrastructure can support JCNR. Preflight checks take place before Cloud-Native Router is installed. Postflight checks verify that your Cloud-Native Router installation is working properly. Postflight checks take place after Cloud-Native Router is installed. See "Cloud-Native Router Readiness Checks" on page 362.
replicas	(Optional) Indicates the number of replicas for cRPD. Default is 1. The value for this key must be specified for multi-node clusters. The value is equal to the number of nodes running JCNR.

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>noLocalSwitching</td><td>Not applicable.</td></tr>
<tr><td>iamrole</td><td>Not applicable.</td></tr>
<tr><td>fabricInterface</td><td>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: Use the L3 only section to configure fabric interfaces for Amazon EKS. The L2 only and L2-L3 sections are not applicable for EKS deployments.

For example:
<pre>
L3 only
- eth1:
    ddp: "off"
- eth2:
    ddp: "off"
</pre></td></tr>
<tr><td>subnet</td><td>An alternative mode of input to interface names. For example:
<pre>
- subnet: 10.40.1.0/24
  gateway: 10.40.1.1
  ddp: "off"
</pre>With the <code>subnet</code> 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 in a multi-node cluster.</td></tr>
<tr><td>ddp</td><td>Not applicable.</td></tr>
<tr><td>interface_mode</td><td>Not applicable.</td></tr>
<tr><td>vlan-id-list</td><td>Not applicable.</td></tr>
</table>

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

Key	Description
storm-control-profile	Not applicable.
native-vlan-id	Not applicable.
no-local-switching	Not applicable.
qosSchedulerProfileName	Specifies the QoS scheduler profile applicable to this interface. See the `qosSchedulerProfiles` section. If you don't specify a profile, then the QoS scheduler is disabled for this interface, which means that packets are scheduled with no regard to traffic class.
fabricWorkloadInterface	Not applicable.
log_level	Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave it set to the default INFO unless instructed to change it by Juniper Networks support.
log_path	The defined directory stores various JCNR-related descriptive logs such as **contrail-vrouter-agent.log**, **contrail-vrouter-dpdk.log**, etc. Default is **/var/log/jcnr/**.
syslog_notifications	Indicates the absolute path to the file that stores `syslog-ng` generated notifications in JSON format. Default is **/var/log/jcnr/jcnr_notifications.json**.
corePattern	Indicates the `core_pattern` for the core file. If left blank, then Cloud-Native Router pods will not overwrite the default pattern on the host. **NOTE**: Set the `core_pattern` on the host before deploying JCNR. You can change the value in **/etc/sysctl.conf**. For example, `kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

Key	Description
coreFilePath	Indicates the path to the `core` file. Default is **/var/crash**.

nodeAffinity	(Optional) Defines labels on nodes to determine where to place the vRouter pods. By default the vRouter pods are deployed to all nodes of a cluster.
	
	In the example below, the node affinity label is defined as `key1=jcnr`. You must apply this label to each node where Cloud-Native Router is to be deployed:
	
	`nodeAffinity:`
	`  - key: key1`
	`    operator: In`
	`    values:`
	`      - jcnr`
	
	NOTE: This key is a global setting.

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 path with the path in your distribution (for example, **/var/opt/cni/bin**).
grpcTelemetryPort	(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50051.
grpcVrouterPort	(Optional) Default is 50052. Configure to override.
vRouterDeployerPort	(Optional) Default is 8081. Configure to override.

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td>cpu_core_mask</td><td> If present, this indicates that you want to use static CPU allocation to allocate cores to the forwarding plane. This value should be a comma-delimited list of isolated CPU cores that you want to statically allocate to the forwarding plane (for example, <code>cpu_core_mask: "2,3,22,23"</code>). Use the cores not used by the host OS. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane. NOTE: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Cloud-Native Router Readiness preflight checks, if enabled, will fail the installation if you specify both. </td></tr>
<tr><td>guaranteedVrouterCpus</td><td> If present, this indicates that you want to use the Kubernetes CPU Manager to allocate CPU cores to the forwarding plane. This value should be the number of guaranteed CPU cores that you want the Kubernetes CPU Manager to allocate to the forwarding plane. You should set this value to at least one more than the number of forwarding cores. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane. NOTE: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Using both can lead to unpredictable behavior. </td></tr>
<tr><td>dpdkCtrlThreadMask</td><td> Specifies the CPU core(s) to allocate to vRouter DPDK control threads when using static CPU allocation. This list should be a subset of the cores listed in <code>cpu_core_mask</code> and can be the same as the list in <code>serviceCoreMask</code>. CPU cores listed in <code>cpu_core_mask</code> but not in <code>serviceCoreMask</code> or <code>dpdkCtrlThreadMask</code> are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane. </td></tr>
</table>

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

Key	Description
serviceCoreMask	Specifies the CPU core(s) to allocate to vRouter DPDK service threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `dpdkCtrlThreadMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.
numServiceCtrlThreadCPU	Specifies the number of CPU cores to allocate to vRouter DPDK service/control traffic when using the Kubernetes CPU Manager. This number should be smaller than the number of `guaranteedVrouterCpus` cores. The remaining `guaranteedVrouterCpus` cores are allocated for forwarding. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
numberOfSchedulerLcores	The number of CPU cores that you want Kubernetes CPU Manager to dedicate to your QoS schedulers. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
restoreInterfaces	Set to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts or if Cloud-Native Router is uninstalled.
bondInterfaceConfigs	Not applicable.
mtu	Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default is 9000.
qosSchedulerProfiles	Defines the QoS scheduler profiles that are referenced from the `fabricInterface` section.
sched_profile_1	The name of the QoS scheduler profile.

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

Key	Description
	cpu Specify the CPU core(s) to dedicate to the scheduler. If `cpu_core_mask` is specified, this should be a unique additional core(s). **bandwidth** Specify the bandwidth in Gbps.
stormControlProfiles	Not applicable.
dpdkCommandAdditionalArgs	Pass any additional DPDK command line parameters. The `--yield_option 0` is set by default and implies the DPDK forwarding cores will not yield their assigned CPU cores. Other common parameters that can be added are `tx` and `rx` descriptors and `mempool`. For example: `dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 2048 --dpdk_rxd_sz 2048 --vr_mempool_sz 131072"` **NOTE**: Changing the number of `tx` and `rx` descriptors and the `mempool` size affects the number of huge pages required. If you make explicit changes to these parameters, set the number of huge pages to 10 (x 1 GB). See "Configure Huge Pages" on page 401 for information on how to configure huge pages.
dpdk_monitoring_thread_config	(Optional) Enables a monitoring thread for the vRouter DPDK container. Every `loggingInterval` seconds, a log containing the information indicated by `loggingMask` is generated.
loggingMask	Specifies the information to be generated. Represented by a bitmask with bit positions as follows: • 0b001 is the nl_counter • 0b010 is the lcore_timestamp • 0b100 is the profile_histogram
loggingInterval	Specifies the log generation interval in seconds.

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

Key	Description
ddp	Not applicable.
twampPort	(Optional) The TWAMP session reflector port (if you want TWAMP sessions to use vRouter timestamps). The vRouter listens to TWAMP test messages on this port and inserts/overwrites timestamps in TWAMP test messages. Timestamping TWAMP messages at the vRouter (instead of at cRPD) leads to more accurate measurements. Valid values are 862 and 49152 through 65535. If this parameter is absent, then the vRouter does not insert or overwrite timestamps in the TWAMP session. Timestamps are taken and inserted by cRPD instead. See *Two-Way Active Measurement Protocol (TWAMP)*.
vrouter_dpdk_uio_driver	The uio driver is `vfio-pci`.
agentModeType	Set to dpdk.
fabricRpfCheckDisable	Set to false to enable the RPF check on all Cloud-Native Router fabric interfaces. By default, RPF check is disabled.
telemetry	(Optional) Configures cRPD telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
disable	Set to true to disable cRPD telemetry. Default is false, which means that cRPD telemetry is enabled by default.
metricsPort	The port that the cRPD telemetry exporter is listening to Prometheus queries on. Default is 8072.
logLevel	One of warn, warning, info, debug, trace, or verbose. Default is info.
gnmi	(Optional) Configures cRPD gNMI settings.

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

Key	Description
	enable Set to true to enable the cRPD telemetry exporter to respond to gNMI requests.
vrouter	
telemetry	(Optional) Configures vRouter telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
	metricsPort Specify the port that the vRouter telemetry exporter listens to Prometheus queries on. Default is 8070.
	logLevel One of warn, warning, info, debug, trace, or verbose. Default is info.
	gnmi (Optional) Configures vRouter gNMI settings. enable - Set to true to enable the vRouter telemetry exporter to respond to gNMI requests.
persistConfig	Set to true if you want Cloud-Native Router pod configuration to persist even after uninstallation. This option can only be set for L2 mode deployments. Default is false.
enableLocalPersistence	Set to true if you're using the cRPD CLI or NETCONF to configure JCNR. When set to true, the cRPD CLI and NETCONF configuration persists through node reboots, cRPD pod restarts, and Cloud-Native Router upgrades. Default is false.
interfaceBoundType	Not applicable.
networkDetails	Not applicable.
networkResources	Not applicable.
contrail-tools	

Table 18: Helm Chart Description for Amazon EKS Deployment *(Continued)*

Key	Description
install	Set to true to install contrail-tools (used for debugging).

Customize Cloud-Native Router Configuration

SUMMARY

Read this topic to understand how to customize Cloud-Native Router configuration using a Configlet custom resource.

IN THIS SECTION

Configlet Custom Resource

Starting with Juniper Cloud-Native Router (JCNR) Release 24.2, we support customizing Cloud-Native Router configuration using a configlet custom resource. The configlet can be generated either by rendering a predefined template of supported Junos configuration or using raw configuration. The generated configuration is validated and deployed on the Cloud-Native Router controller (cRPD) as one or more Junos configuration groups.

> **NOTE**: You can configure Cloud-Native Router using either configlets or the cRPD CLI or NETCONF. If you use the cRPD CLI or NETCONF, be sure to enable local persistence in **values.yaml** (`enableLocalPersistence: true`) so that your CLI or NETCONF configuration persists across reboots and upgrades.

NOTE: Using both configlets and the cRPD CLI or NETCONF to configure Cloud-Native Router may lead to unpredictable behavior. Use one or the other, but not both.

Configuration Examples

You create a configlet custom resource of the kind `Configlet` in the `jcnr` namespace. You provide raw configuration as Junos `set` commands.

Use `crpdSelector` to control where the configlet applies. The generated configuration is deployed to cRPD pods on nodes matching the specified label only. If `crpdSelector` is not defined, the configuration is applied to all cRPD pods in the cluster.

An example configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample             # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker                   # <-- Node label to select the cRPD pods
```

You can also use a templatized configlet yaml that contains keys or variables. The values for variables are provided by a configletDataValue custom resource, referenced by `configletDataValueRef` . An example templatized configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-with-template      # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
```

```
  crpdSelector:
    matchLabels:
      node: worker                           # <-- Node label to select the cRPD pods
  configletDataValueRef:
    name: "configletdatavalue-sample"        # <-- Configlet Data Value resource name
```

To render configuration using the template, you must provide key:value pairs in the ConfigletDataValue custom resource:

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"           # <-- Key:Value pair
  }
```

The generated configuration is validated and applied to all or selected cRPD pods as a Junos Configuration Group.

Applying the Configlet Resource

The configlet resource can be used to apply configuration to selected or all cRPD pods either when Cloud-Native Router is deployed or once the cRPD pods are up and running. Let us look at configlet deployment in detail.

Applying raw configuration

1. Create raw configuration configlet yaml. The example below configures a loopback interface in cRPD.

```
cat configlet-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
```

```
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker
```

2. Apply the configuration using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                                  AGE
configlet-sample-node1                10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

```
kubectl describe nodeconfiglet configlet-sample-node1 -n jcnr
```

   The following output has been trimmed for brevity:

```
Name:         configlet-sample-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
Annotations:  <none>
```

```
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 10.10.10.1/32;
            }
        }
    }
}
```

NOTE: The configuration generated using configlets is applied to cRPD as configuration groups. We therefore recommend that you not use configuration groups when specifying your configlet.

Applying templatized configuration

1. Create the templatized configlet yaml and the configlet data value yaml for key:value pairs.

```
cat configlet-sample-template.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-template
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: master
  configletDataValueRef:
    name: "configletdatavalue-sample"
```

```
cat configletdatavalue-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"
  }
```

2. Apply the configuration using the `kubectl apply` command, starting with the config data value yaml.

```
kubectl apply -f configletdatavalue-sample.yaml
```

```
configletdatavalue.configplane.juniper.net/configletdatavalue-sample created
```

```
kubectl apply -f configlet-sample-template.yaml
```

```
configlet.configplane.juniper.net/configlet-sample-template created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                                           AGE
configlet-sample-template-node1                10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

```
kubectl describe nodeconfiglet configlet-sample-template-node1 -n jcnr
```

   The following output has been trimmed for brevity:

```
Name:         configlet-sample-template-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
Annotations:  <none>
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
```

```
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample-template
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample-template
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32;
            }
        }
    }
}
```

Modifying the Configlet

You can modify a configlet resource by changing the yaml file and reapplying it using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample configured
```

Any changes to existing configlet resource are reconciled by replacing the configuration group on cRPD.

You can delete the configuration group by deleting the configlet resource using the `kubectl delete` command.

```
kubectl delete configlet configlet-sample -n jcnr
```

```
configlet.configplane.juniper.net "configlet-sample" deleted
```

Troubleshooting

If you run into problems, check the `contrail-k8s-deployer` logs. For example:

```
kubectl logs contrail-k8s-deployer-8ff895cc5-cbfwm -n contrail-deploy
```

Cloud-Native Router Operator Service Module: VPC Gateway

SUMMARY

The Cloud-Native Router Operator Service Module is an operator framework that we use to develop cRPD applications and solutions. This section describes how to use the Service Module to implement a VPC gateway between your Amazon EKS cluster and your on-premises Kubernetes cluster.

IN THIS SECTION

Cloud-Native Router VPC Gateway Overview

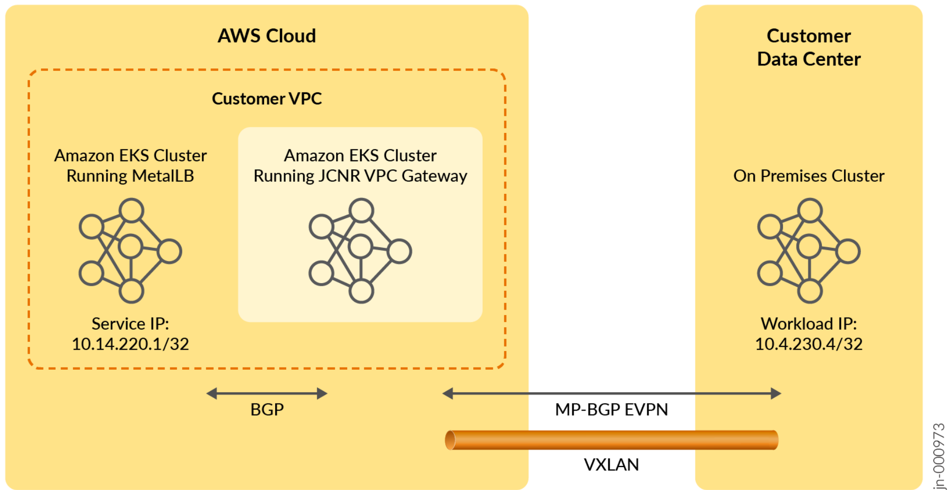
We provide the Cloud-Native Router Operator Service Module to install JCNR (with a BYOL license) on an Amazon EKS cluster and to configure it to act as an EVPN-VXLAN VPC Gateway between a separate Amazon EKS cluster running MetalLB and an on-premises Kubernetes cluster (Figure 4 on page 178).

Once you configure the VPC Gateway custom resource with information on your MetalLB cluster and your on-premises Kubernetes cluster, the VPC Gateway establishes a BGP session with your MetalLB cluster and establishes a BGP EVPN session with your on-premises Kubernetes cluster. Routes learned from the MetalLB cluster are re-advertised to the on-premises cluster using EVPN Type 5 routes. Routes learned from the on-premises cluster are leaked into the route tables of the routing instance for the MetalLB cluster.

The configuration example we'll use in this section connects workloads at 10.4.230.4/32 in the on-premises cluster to services at 10.14.220.1/32 in the MetalLB cluster.

NOTE: Configuring the connectivity between the AWS Cloud and the Customer Data Center is not covered in this procedure. Use your preferred AWS method for connectivity.

Figure 4: Cloud-Native Router VPC Gateway



NOTE: The VPC Gateway custom resource automatically installs Cloud-Native Router with a configuration that is specific to this application. You don't need to install Cloud-Native Router explicitly and you don't need to configure the Cloud-Native Router installation Helm chart.

Install the Cloud-Native Router VPC Gateway

This is the main procedure. Start here.

1. Prepare the clusters.

   a. Prepare the Cloud-Native Router VPC Gateway cluster. See "Prepare the Cloud-Native Router VPC Gateway Cluster" on page 193.

   b. Prepare the MetalLB cluster. See "Prepare the MetalLB Cluster" on page 190.

   c. Prepare the on-premises cluster. See "Prepare the On-Premises Cluster" on page 195

   After preparing the clusters, you can start installation of the Cloud-Native Router VPC Gateway. Execute the remaining steps in the Cloud-Native Router VPC Gateway cluster.

2. Download and install the Cloud-Native Router Service Module Helm chart on the cluster.

   You can download the Cloud-Native Router Service Module Helm chart from the Juniper Networks software download site. See "Cloud-Native Router Software Download Packages" on page 387.

3. Install the downloaded Helm chart.

```
helm install vpcgwy Juniper_Cloud_Native_Router_Service_Module_<release>.tgz
```

NOTE: The provided Helm chart installs the Cloud-Native Router VPC Gateway on cores 2, 3, 22, and 23. Therefore ensure that the nodes in your cluster have at least 24 cores and that the specified cores are free to use.

Check that the controller-manager and the contrail-k8s-deployer pods are up.

```
kubectl get pods -A
```

```
NAMESPACE         NAME                                       READY   STATUS
svcmodule-system  controller-manager-67898d794d-4cpsw        2/2     Running
cert-manager      cert-manager-5bd57786d4-mf7hq              1/1     Running
cert-manager      cert-manager-cainjector-57657d5754-5d2xc   1/1     Running
cert-manager      cert-manager-webhook-7d9f8748d4-p482n      1/1     Running
contrail-deploy   contrail-k8s-deployer-546587dcbc-bjbrg     1/1     Running
kube-system       aws-node-dhsgv                             2/2     Running
kube-system       aws-node-n6kcx                             2/2     Running
kube-system       coredns-54d6f577c6-m7q8h                   1/1     Running
kube-system       coredns-54d6f577c6-qc76c                   1/1     Running
kube-system       eks-pod-identity-agent-6k6xj               1/1     Running
kube-system       eks-pod-identity-agent-rvqz7               1/1     Running
kube-system       kube-proxy-nqpsd                           1/1     Running
kube-system       kube-proxy-vzbnv                           1/1     Running
```

4. Configure the Cloud-Native Router VPC Gateway custom resource.

   This custom resource contains information on the MetalLB cluster and the on-premises cluster.

   a. Create a YAML file that contains the desired configuration. We'll put our Cloud-Native Router VPC Gateway pods into a namespace that we'll call `gateway`.

      The YAML file has the following format:

```
apiVersion: v1
kind: Namespace
metadata:
  name: gateway
```

```
---
apiVersion: workflow.svcmodule.juniper.net/v1
kind: VpcGateway
metadata:
  name: vpc-gw
  namespace: gateway
spec:
  <see table>
```

Table 19 on page 180 describes the main configuration fields for the `spec` section. In the `spec` definition, `application` refers to the MetalLB cluster and `client` refers to the on-premises cluster.

Table 19: Spec Descriptions

Spec Field	Description
applicationTopology	This section contains information on the MetalLB cluster.
applicationInterface	The name of the interface connecting to the MetalLB cluster.
bgpSpeakerType	Specify metallb when connecting to the MetalLB cluster.
clusters	
kubeconfigSecretName	The secret containing the kubeconfig of the MetalLB cluster.
name	The name of the MetalLB cluster.
enableV6	(Optional) True or false. Enables or disables IPv6 in the MetalLB cluster. Default is false.

Table 19: Spec Descriptions *(Continued)*

Spec Field	Description
neighbourDiscovery	(Optional) True or false. Governs how BGP neighbors (BGP speakers from the MetalLB cluster) are determined. When set to true, BGP neighbors with addresses specified in sessionPrefix or with addresses in the application interface's subnet are accepted. When set to false, the remote MetalLB cluster's cRPD pod IP is used as the BGP neighbor. Default is false.
routePolicyOverride	(Optional) True or false. When set to true, a route policy called "export-onprem" is used to govern what MetalLB cluster routes are exported to the on-premises cluster. This gives you the opportunity to create your own export policy. You must create this policy manually and call it "export-onprem". Default is false, which means that all MetalLB cluster routes are exported to the on-premises cluster.
sessionPrefix	(Optional) Used when neighbourDiscovery is set to true. When present, it indicates the CIDR from which BGP sessions from the MetalLB cluster are accepted. Default is to accept BGP sessions from BGP neighbors in the application interface's subnet.
client	Information related to the on-premises cluster.
address	The BGP speaker IP address of the on-premises cluster. The Cloud-Native Router VPC Gateway establishes a direct eBGP session with this address. This eBGP session is used to learn the route to the loopback address, which is used to establish the subsequent BGP EVPN session.

Table 19: Spec Descriptions *(Continued)*

Spec Field	Description
asn	The AS number of the eBGP speaker in the client cluster. The Cloud-Native Router VPC Gateway validates this when establishing the direct eBGP session with the BGP speaker in the on-premises cluster.
loopbackAddress	The loopback address of the BGP speaker in the on-premises cluster. The Cloud-Native Router VPC Gateway uses this IP address to establish a BGP EVPN session with the BGP speaker in the on-premises cluster.
myASN	The local AS number that the Cloud-Native Router VPC Gateway uses for the direct eBGP session with the BGP speaker in the on-premises cluster.
routeTarget	The route target for the EVPN routes in the on-premises cluster.
vrrp	Always set to true. This enables VRRP on interfaces towards the on-premises cluster.
clientInterface	The name of the interface connecting to the on-premises cluster.
dpdkDriver	Set to vfio-pci.
loopbackIPPool	The IP address pool used for assigning IP addresses to the cRPD instances created in the cluster (in CIDR format). **NOTE**: The number of addresses in the pool must be at least one more than the number of replicas.

Table 19: Spec Descriptions *(Continued)*

Spec Field	Description
nodeSelector	(Optional) Used in conjunction with a node's labels to determine whether the VPC Gateway pod can run on a node. This selector must match a node's labels for the pod to be scheduled on that node.
replicas	(Optional) The number of JCNRs created. Default is 1.

NOTE: Armed with the MetalLB kubeconfig, the Cloud-Native Router VPC Gateway has sufficient information to configure BGP sessions automatically with the MetalLB cluster. You don't need to provide any parameters other than what's listed in the table.

Here's an example of a working configuration:

```
apiVersion: v1
kind: Namespace
metadata:
  name: jcnr-gateway
---
apiVersion: workflow.svcmodule.juniper.net/v1
kind: VpcGateway
metadata:
  name: vpc-gw
  namespace: gateway
spec:
  dpdkDriver: vfio-pci
  replicas: 1
  clientInterface: eth3
  loopbackIPPool: 10.14.140.0/28
  applicationTopology:
    applicationInterface: eth2
    bgpSpeakerType: metallb
    clusters:
```

```
      - name: metallb-1
        kubeconfigSecretName: metallb-cluster-kubeconfig
  client:
    asn: 65010
    myASN: 65000
    address: 10.14.205.158
    loopbackAddress: 10.14.140.200
    routeTarget: target-1-4
    vrrp: true
```

b. Apply the YAML file to the cluster.

```
kubectl apply -f vpcGateway.yaml
```

where **vpcGateway.yaml** is the YAML file defining the Cloud-Native Router VPC Gateway.

c. Check the pods.

```
kubectl get pods -A
```

```
NAMESPACE         NAME                                                  READY   STATUS
svcmodule-system  controller-manager-67898d794d-4cpsw                   2/2
Running
cert-manager      cert-manager-5bd57786d4-mf7hq                         1/1
Running
cert-manager      cert-manager-cainjector-57657d5754-5d2xc              1/1
Running
cert-manager      cert-manager-webhook-7d9f8748d4-p482n                 1/1
Running
contrail-deploy   contrail-k8s-deployer-546587dcbc-bjbrg                1/1
Running
contrail          vpc-gw-crpdgroup-0-x-contrail-vrouter-nodes-s9wkk     2/2
Running
contrail          vpc-gw-crpdgroup-0-x-contrail-vrouter-nodes-vrdpdk-jczh5  1/1
Running
jcnr              jcnr-gateway-vpc-gw-crpdgroup-0-0                     2/2
Running
kube-system       aws-node-dhsgv                                        2/2
Running
kube-system       aws-node-n6kcx                                        2/2
```

```
Running
kube-system      coredns-54d6f577c6-m7q8h                         1/1
Running
kube-system      coredns-54d6f577c6-qc76c                         1/1
Running
kube-system      eks-pod-identity-agent-6k6xj                     1/1
Running
kube-system      eks-pod-identity-agent-rvqz7                     1/1
Running
kube-system      kube-proxy-nqpsd                                 1/1
Running
kube-system      kube-proxy-vzbnv                                 1/1
Running
```

5. Verify your installation.

   Find the name of the configlet:

```
kubectl get nodeconfiglet -n jcnr
```

```
NAME                 AGE
vpc-gw-crpdgroup-0   8h
```

   See how the configlet is configured. For example:

```
kubectl describe nodeconfiglet -n jcnr vpc-gw-crpdgroup-0
```

```
Name:          vpc-gw-crpdgroup-0
Namespace:     jcnr
Labels:        core.juniper.net/nodeName=ip-10-75-66-162.us-west-2.compute.internal
Annotations:   <none>
API Version:   configplane.juniper.net/v1
Kind:          NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-24T23:32:35Z
  Finalizers:
    node-configlet.finalizers.deployer.juniper.net
  Generation:  26
  Managed Fields:
    API Version:  configplane.juniper.net/v1
```

```
    Fields Type:  FieldsV1
    fieldsV1:
      f:status:
        .:
        f:message:
        f:status:
    Manager:      manager
    Operation:    Update
    Subresource:  status
    Time:         2024-06-24T23:32:36Z
    API Version:  configplane.juniper.net/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:finalizers:
          .:
          v:"node-configlet.finalizers.deployer.juniper.net":
        f:ownerReferences:
          .:
          k:{"uid":"00c67217-87e7-434d-8d6a-8256f2d9d206"}:
      f:spec:
        .:
        f:clis:
        f:nodeName:
    Manager:    manager
    Operation:  Update
    Time:       2024-06-25T02:22:26Z
  Owner References:
    API Version:           configplane.juniper.net/v1
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  JcnrInstance
    Name:                  vpc-gw-crpdgroup-0
    UID:                   00c67217-87e7-434d-8d6a-8256f2d9d206
  Resource Version:        133907
  UID:                     340a19d0-9de5-414d-b2ac-c3831203877c
Spec:
  Clis:
    set interfaces eth2 unit 0 family inet address 10.14.207.30/22
    set interfaces eth2 mac 52:54:00:a4:c3:85
    set interfaces eth2 mtu 9216
    set interfaces eth3 unit 0 family inet address 10.14.205.159/22
    set interfaces eth3 mac 52:54:00:ee:4b:3f
```

```
    set interfaces eth3 mtu 9216
    set interfaces lo0 unit 0 family inet address 10.14.140.1/32
    set interfaces lo0 mtu 9216
    set policy-options policy-statement default-rt-to-aws-export then reject
    set policy-options policy-statement default-rt-to-aws-export term awsv4 from family inet
    set policy-options policy-statement default-rt-to-aws-export term awsv4 from protocol evpn
    set policy-options policy-statement default-rt-to-aws-export term awsv4 then accept
    set policy-options policy-statement default-rt-to-aws-export term awsv6 from family inet6
    set policy-options policy-statement default-rt-to-aws-export term awsv6 from protocol evpn
    set policy-options policy-statement default-rt-to-aws-export term awsv6 then accept
    set policy-options policy-statement export-direct then reject
    set policy-options policy-statement export-direct term directly-connected from protocol
direct
    set policy-options policy-statement export-direct term directly-connected then accept
    set policy-options policy-statement export-evpn then reject
    set policy-options policy-statement export-evpn term evpn-connected from protocol evpn
    set policy-options policy-statement export-evpn term evpn-connected then accept
    set policy-options policy-statement export-onprem then reject
    set policy-options policy-statement export-onprem term learned-from-bgp from protocol bgp
    set policy-options policy-statement export-onprem term learned-from-bgp then accept
    set routing-instances application-ri protocols bgp group vpc-gw-application local-address
10.14.207.30
    set routing-instances application-ri protocols bgp group vpc-gw-application export export-
evpn
    set routing-instances application-ri protocols bgp group vpc-gw-application peer-as 64513
    set routing-instances application-ri protocols bgp group vpc-gw-application local-as 64512
    set routing-instances application-ri protocols bgp group vpc-gw-application multihop
    set routing-instances application-ri protocols bgp group vpc-gw-application allow
10.14.207.29/22
    set routing-instances application-ri protocols evpn ip-prefix-routes advertise direct-
nexthop
    set routing-instances application-ri protocols evpn ip-prefix-routes encapsulation vxlan
    set routing-instances application-ri protocols evpn ip-prefix-routes vni 4096
    set routing-instances application-ri protocols evpn ip-prefix-routes export export-onprem
    set routing-instances application-ri protocols evpn ip-prefix-routes route-attributes
community export-action allow
    set routing-instances application-ri protocols evpn ip-prefix-routes route-attributes
community import-action allow
    set routing-instances application-ri interface eth2
    set routing-instances application-ri vrf-target target:1:4
    set routing-instances application-ri instance-type vrf
    set routing-options route-distinguisher-id 10.14.140.1
    set routing-options router-id 10.14.140.1
```

```
    set protocols bgp group vpc-gw-client-lo local-address 10.14.140.1
    set protocols bgp group vpc-gw-client-lo peer-as 64512
    set protocols bgp group vpc-gw-client-lo local-as 64512
    set protocols bgp group vpc-gw-client-lo family evpn signaling
    set protocols bgp group vpc-gw-client-lo neighbor 10.14.140.200
    set protocols bgp group vpc-gw-client-direct export export-direct
    set protocols bgp group vpc-gw-client-direct peer-as 65010
    set protocols bgp group vpc-gw-client-direct local-as 65000
    set protocols bgp group vpc-gw-client-direct multihop
    set protocols bgp group vpc-gw-client-direct neighbor 10.14.205.158
  Node Name:  ip-10-75-66-162.us-west-2.compute.internal
Status:
  Message:  Configuration committed
  Status:   True
Events:     <none>
```

6. Verify your installation.

   a. Access the cRPD pod.

   ```
   kubectl exec -n jcnr jcnr-gateway-vpc-gw-crpdgroup-0-0 -c crpd -it -- sh
   ```

   b. Enter CLI mode.

   ```
   cli
   ```

   c. Check the BGP peers.

   ```
   show bgp summary
   Threading mode: BGP I/O
   Default eBGP mode: advertise - accept, receive - accept
   Groups: 3 Peers: 3 Down peers: 0
   Unconfigured peers: 1
   Table          Tot Paths  Act Paths Suppressed    History Damp State    Pending
   bgp.evpn.0
                          2          2          0          0          0          0
   inet.0
                          4          1          0          0          0          0
   Peer                     AS      InPkt     OutPkt    OutQ   Flaps Last Up/Dwn State|
   #Active/Received/Accepted/Damped...
   10.14.140.200         64512       6514       6471       0       1  2d 0:49:34 Establ
   ```

```
  bgp.evpn.0: 2/2/2/0
  application-ri.evpn.0: 2/2/2/0
10.14.205.158         65010       6386       6363       0       3  2d 0:01:56 Establ
  inet.0: 1/4/4/0
10.14.207.29          64513       5758       6352       0       0 1d 23:56:40 Establ
  application-ri.inet.0: 1/1/1/0
```

In the output above, the Cloud-Native Router VPC Gateway has the following BGP sessions:

- with the iBGP speaker in the on-premises cluster at 10.14.140.200 for EVPN routes
- with the eBGP speaker in the on-premises cluster at 10.14.205.158 for the direct eBGP session
- with the MetalLB cluster at 10.14.207.29

d. Check the routes to the MetalLB cluster and the on-premises cluster.

Check the route to the Nginx service in the MetalLB cluster:

```
show route 10.14.220.1

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

10.14.220.1/32     *[BGP/170] 1d 00:24:25, localpref 100
                      AS path: 64513 I, validation-state: unverified
                    >  to 10.14.207.29 via eth2
```

Check the route to the workloads in the on-premises cluster:

```
show route 10.4.230.4

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

10.4.230.4/32       *[EVPN/170] 1d 17:51:00
                    >  to 10.14.205.158 via eth3
```

With the routes successfully exchanged, the on-premises workloads at 10.4.230.4 can access the MetalLB cluster at 10.14.220.1.

Prepare the MetalLB Cluster

The MetalLB cluster is the Amazon EKS cluster that you ultimately want to connect to your on-premises cluster. Follow this procedure to prepare your MetalLB cluster to establish a BGP session with the Cloud-Native Router VPC Gateway.

1. Create the Amazon EKS cluster where you'll be running the MetalLB service.
2. Deploy MetalLB on that cluster. MetalLB provides a network load balancer implementation for your cluster.

   See https://metallb.universe.tf/configuration/ for information on deploying MetalLB.
3. Create the necessary MetalLB resources. As a minimum, you need to create the MetalLB IPAddressPool resource and the MetalLB BGPAdvertisement resource.

   a. Create the MetalLB IPAddressPool resource.

      Here's an example of a YAML file that defines the IPAddressPool resource.

      ```
      apiVersion: metallb.io/v1beta1
      kind: IPAddressPool
      metadata:
        name: first-pool
        namespace: metallb-system
      spec:
        addresses:
        - 10.14.220.0/24
        avoidBuggyIPs: true
      ```

      In this example, MetalLB will assign load balancer IP addresses from the 10.14.220.0/24 range.

      Apply the above YAML to the cluster to create the IPAddressPool.

      ```
      kubectl apply -f ipaddresspool.yaml
      ```

      where **ipaddresspool.yaml** is the name of the YAML file defining the IPAddressPool resource.

   b. Create the MetalLB BGPAdvertisement resource.

      Here's an example of a YAML file that defines the BGPAdvertisement resource.

      ```
      apiVersion: metallb.io/v1beta1
      kind: BGPAdvertisement
      metadata:
      ```

```
  name: example
  namespace: metallb-system
```

The BGPAdvertisement resource advertises your service IP addresses to external routers (for example, to your Cloud-Native Router VPC Gateway).

Apply the above YAML to the cluster to create the BGPAdvertisement resource.

```
kubectl apply -f bgpadvertisement.yaml
```

where **bgpadvertisement.yaml** is the name of the YAML file defining the BGPAdvertisement resource.

4. Create the LoadBalancer service. The LoadBalancer service provides the entry point for external workloads to reach the cluster. You can create any LoadBalancer service of your choice.

   Here's an example YAML for an Nginx LoadBalancer service.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: <image repo URL>
        ports:
        - name: http
          containerPort: 80

---
apiVersion: v1
kind: Service
metadata:
  name: nginx
```

```
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: nginx
  type: LoadBalancer
```

Apply the above YAML to the cluster to create the Nginx LoadBalancer service.

```
kubectl apply -f nginx.yaml
```

where **nginx.yaml** is the name of the YAML file defining the Nginx service.

5. Verify your installation.

   a. Take a look at the pods in your cluster.

      For example:

      ```
      kubectl get pods -A
      ```

      ```
      NAMESPACE        NAME                          READY   STATUS    RESTARTS       AGE
      default          nginx-6d66d85dc4-h6dng        1/1     Running   0              9d
      kube-system      aws-node-vdhv9                2/2     Running   2 (28d ago)    28d
      kube-system      coredns-54d6f577c6-lbznn      1/1     Running   1 (28d ago)    29d
      kube-system      coredns-54d6f577c6-stljk      1/1     Running   1 (28d ago)    29d
      kube-system      eks-pod-identity-agent-kqtcb  1/1     Running   1 (28d ago)    28d
      kube-system      kube-proxy-fxcjq              1/1     Running   1 (28d ago)    28d
      metallb-system   controller-5c6b6c8447-2jdzc   1/1     Running   0              28d
      metallb-system   speaker-xhkpd                 1/1     Running   0              28d
      ```

      The example output shows that both MetalLB and Nginx are up.

   b. Check the assigned external IP address for the Nginx service.

For example:

```
kubectl get svc nginx
```

```
NAME    TYPE          CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
nginx   LoadBalancer  10.100.65.169  10.14.220.1   80:30623/TCP   9d
```

In this example, MetalLB has assigned 10.14.220.1 to the Nginx LoadBalancer service. This is the overlay IP address that workloads in the on-premises cluster can use to reach services in the MetalLB cluster.

Prepare the Cloud-Native Router VPC Gateway Cluster

1. Create the Amazon EKS cluster that you want to act as the Cloud-Native Router VPC Gateway.

   The cluster must meet the system requirements described in "System Requirements for EKS Deployment" on page 149.

   Since you're not installing Cloud-Native Router explicitly, you can ignore any requirement that relates to downloading the Cloud-Native Router software package or configuring the Cloud-Native Router Helm chart.
2. Ensure all worker nodes in the cluster have identical interface names and identical root passwords.

   In this example, we'll use eth2 to connect to the MetalLB cluster and eth3 to connect to the on-premises cluster.
3. Once the cluster is up, create a **jcnr-secrets.yaml** file with the below contents.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: jcnr
---
apiVersion: v1
kind: Secret
metadata:
  name: jcnr-secrets
  namespace: jcnr
data:
  root-password: <add your password in base64 format>
```

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

4. Follow the steps in "Installing Your License" on page 353 to install your Cloud-Native Router BYOL license in the **jcnr-secrets.yaml** file.
5. Enter the base64-encoded form of the root password for your nodes into the **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 the designated location in **jcnr-secrets.yaml**.

6. Apply **jcnr-secrets.yaml** to the cluster.

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

7. Create the secret for accessing the MetalLB cluster.

   a. Base64-encode the MetalLB cluster kubeconfig file.

   ```
   base64 -w0 <metalLB-kubeconfig>
   ```

   where `<metalLB-kubeconfig>` is the kubeconfig file for the MetalLB cluster.

   The output of this command is the base64-encoded form of the MetalLB cluster kubeconfig.

   b. Create the YAML defining the MetalLB cluster kubeconfig secret. We'll use a namespace called `jcnr-gateway`, which we'll define later.

   ```
   apiVersion: v1
   data:
     kubeconfig: |-
   <base64-encoded kubeconfig of MetalLB cluster>
   kind: Secret
   metadata:
     name: metallb-cluster-kubeconfig
   ```

```
  namespace: jcnr-gateway
type: Opaque
```

where *<base64-encoded kubeconfig of MetalLB cluster>* is the base64-encoded output from the previous step.

c. Apply the YAML.

```
kubectl apply -f metallb-cluster-kubeconfig-secret.yaml
```

where **metallb-cluster-kubeconfig-secret.yaml** is the name of the YAML file defining the secret.

8. Install webhooks.

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.12.0/cert-manager.yaml
```

9. Create the jcnr-aws-configmap. See "Cloud-Native Router ConfigMap for VRRP" on page 153.

Your cluster is now ready for you to install the Cloud-Native Router VPC Gateway, but let's prepare the on-premises cluster first.

Prepare the On-Premises Cluster

The Cloud-Native Router VPC Gateway sets up an eBGP session and an iBGP session with the on-premises cluster:

- The Cloud-Native Router VPC Gateway uses the eBGP session to learn the loopback IP address of the BGP speaker in the on-premises cluster. The Cloud-Native Router VPC Gateway then uses the loopback IP address to establish the subsequent iBGP session.
- The Cloud-Native Router VPC Gateway uses the iBGP session to learn routes to the workloads in the on-premises cluster. For the iBGP session, you must configure the local and peer AS number to be 64512.

The Cloud-Native Router VPC Gateway does not impose any restrictions on the on-premises cluster as long as you configure it to establish the BGP sessions with the Cloud-Native Router VPC Gateway as described above and to expose routes to the desired workloads.

We don't cover configuring the on-premises cluster because that's very device-specific. You should configure the following, however, in order to be consistent with our ongoing example:

- an eBGP speaker at 10.14.205.158 for the eBGP session

- an iBGP speaker at 10.14.140.200 for exchanging EVPN routes
- workloads reachable at 10.4.230.4/32

5

CHAPTER

Install Cloud-Native Router on Google Cloud Platform

IN THIS CHAPTER

Install and Verify Juniper Cloud-Native Router for GCP Deployment

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 Juniper Support Site Package

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

1. Review the "System Requirements for GCP Deployment" on page 208 section to ensure the setup has all the required configuration.
2. Download the desired Cloud-Native Router software package to the directory of your choice.

   You have the option of downloading the package to install Cloud-Native Router only or downloading the package to install JNCR together with Juniper cSRX. See "Cloud-Native Router Software Download Packages" on page 387 for a description of the packages available. If you don't want to install Juniper cSRX now, you can always choose to install Juniper cSRX on your working Cloud-Native Router installation later.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

   ```
   tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
   ```
4. Change directory to the main installation directory.

   - If you're installing Cloud-Native Router only, then:

     ```
     cd Juniper_Cloud_Native_Router_<release>
     ```

This directory contains the Helm chart for Cloud-Native Router only.

- If you're installing Cloud-Native Router and cSRX at the same time, then:

```
cd Juniper_Cloud_Native_Router_CSRX_<release>
```

This directory contains the combination Helm chart for Cloud-Native Router and cSRX.

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

5. View the contents in the current directory.

```
ls
helmchart  images  README.md  secrets
```

6. Change to the **helmchart** directory and expand the Helm chart.

```
cd helmchart
```

- For Cloud-Native Router only:

```
ls
jcnr-<release>.tgz
```

```
tar -xzvf jcnr-<release>.tgz
```

```
ls
jcnr   jcnr-<release>.tgz
```

The Helm chart is located in the **jcnr** directory.

- For the combined Cloud-Native Router and cSRX:

```
ls
jcnr_csrx-<release>.tgz
```

```
tar -xzvf jcnr_csrx-<release>.tgz
```

```
ls
jcnr_csrx   jcnr_csrx-<release>.tgz
```

  The Helm chart is located in the **jcnr_csrx** directory.

7. 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 398 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 399 for instructions on how to import images to the local container runtime.

8. Follow the steps in "Installing Your License" on page 353 to install your Cloud-Native Router license.

9. 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**.

10. Apply **secrets/jcnr-secrets.yaml** to the cluster.

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

11. If desired, configure how cores are assigned to the vRouter DPDK containers. See "Allocate CPUs to the Cloud-Native Router Forwarding Plane" on page 355.
12. Customize the Helm chart for your deployment using the **helmchart/jcnr/values.yaml** or **helmchart/jcnr_csrx/values.yaml** file.

    See "Customize Cloud-Native Router Helm Chart for GCP Deployment" on page 219 for descriptions of the Helm chart configurations.
13. Optionally, customize Cloud-Native Router configuration.

    See, "Customize Cloud-Native Router Configuration " on page 62 for creating and applying the cRPD customizations.
14. If you're installing Juniper cSRX now, then follow the procedure in "Apply the cSRX License and Configure cSRX" on page 338.
15. Label the nodes where you want Cloud-Native Router to be installed based on the `nodeaffinity` configuration (if defined in the `values.yaml`). For example:

```
kubectl label nodes ip-10.0.100.17.lab.net key1=jcnr --overwrite
```

16. Deploy the Juniper Cloud-Native Router using the Helm chart.

    Navigate to the **helmchart/jcnr** or the **helmchart/jcnr_csrx** directory and run the following command:

```
helm install jcnr .
```

or

```
helm install jcnr-csrx .
```

```
NAME: jcnr
LAST DEPLOYED: Fri Dec 22 06:04:33 2023
NAMESPACE: default
STATUS: deployed
```

```
REVISION: 1
TEST SUITE: None
```

17. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

Sample output:

```
NAME        NAMESPACE    REVISION       UPDATED
STATUS          CHART                     APP VERSION
jcnr        default      1              2023-09-22 06:04:33.144611017 -0400 EDT
deployed        jcnr-<version>                 <version>
```

Install Juniper Cloud-Native Router Via Google Cloud Marketplace

Read this section to learn the steps required to deploy the cloud-native router.

1. Launch the Juniper Cloud-Native Router (PAYG) deployment wizard from the Google Cloud Marketplace.
2. The table below lists the settings to be configured:

Settings	Value
Deployment name	Name of your deployment.
Zone	GCP zone.
Series	N2
Machine Type	n2-standard-32 (32 vCPU, 16 core, 128 GB)
SSH-Keys	SSH key pair for Compute Engine virtual machine (VM) instances.

(Continued)

Settings	Value
Cloud-Native Router License	Base64 encoded license key. To encode the license, copy the license key into a file on your host server and issue the command: `base64 -w 0 licenseFile` Copy and paste the base64 encoded license key in the Cloud-Native Router license field.
cRPD Config Template	Create a config template to customize Cloud-Native Router configuration. See "Customize Cloud-Native Router Configuration " on page 62 for sample cRPD template. The config template must be saved in the GCP bucket as an object. Provide the gsutil URI for the object in the cRPD Config Template field.
cRPD Config Map	Create a config template to customize Cloud-Native Router configuration. See, "Customize Cloud-Native Router Configuration " on page 62 for sample cRPD config map. The config template must be saved in the GCP bucket as an object. Provide the gsutil URI for the object in the cRPD Config Map field.
Boot disk type	Standard Persistent Disk
Boot disk size in GB	50
Network Interfaces	Define additional network interface. An interface in the VPC network is available by default.

3. Review the "System Requirements for GCP Deployment" on page 208 section for additional minimum system requirements. Please note that the settings are pre-configured for the Cloud-Native Router deployment via Google Cloud Marketplace.
4. Click `Deploy` to complete the Cloud-Native Router deployment.

5. Once deployed, you can customize the Cloud-Native Router helm chart. Review the "Customize Cloud-Native Router Helm Chart for GCP Deployment" on page 219 topic for more information. Once configured issue the `helm upgrade` command to deploy the customizations.

```
helm upgrade jcnr .
Release "jcnr" has been upgraded. Happy Helming!
NAME: jcnr
LAST DEPLOYED: Thu Dec 21 03:58:28 2023
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

Verify Installation

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

NOTE: The output shown in this example procedure is affected by the number of nodes in the cluster. The output you see in your setup may differ in that regard.

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         jcnr-0-dp-contrail-vrouter-nodes-b2jxp         2/2     Running   0          87s
contrail         jcnr-0-dp-contrail-vrouter-nodes-vrdpdk-g7wrk  1/1     Running   0          87s
jcnr             jcnr-0-crpd-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   0          4h2m
kube-system      calico-node-28w98                              1/1     Running   0          86d
kube-system      coredns-54bf8d85c7-vkpgs                       1/1     Running   0          3h8m
```

```
kube-system      dns-autoscaler-7944dc7978-ws9fn             1/1    Running   0             86d
kube-system      kube-apiserver-ix-esx-06                    1/1    Running   0             86d
kube-system      kube-controller-manager-ix-esx-06           1/1    Running   0             86d
kube-system      kube-multus-ds-amd64-jl69w                  1/1    Running   0             86d
kube-system      kube-proxy-qm5bl                            1/1    Running   0             86d
kube-system      kube-scheduler-ix-esx-06                    1/1    Running   0             86d
kube-system      nodelocaldns-bntfp                          1/1    Running   0             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      jcnr-0-dp-contrail-vrouter-nodes          1         1         1       1            1
<none>                     90m
contrail      jcnr-0-dp-contrail-vrouter-nodes-vrdpdk   1         1         1       1            1
<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        jcnr-0-crpd   1/1     27m
```

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

   a. View the *Access cRPD CLI* 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 *Access vRouter CLI* 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.

System Requirements for GCP Deployment

IN THIS SECTION

Read this section to understand the system, resource, port, and licensing requirements for installing Juniper Cloud-Native Router on Google Cloud Platform (GCP).

Minimum Host System Requirements for GCP Deployment

Table 20 on page 209 lists the host system requirements for installing Cloud-Native Router on GCP.

NOTE: The settings below are pre-configured when you deploy Cloud-Native Router via the Google Cloud Marketplace.

Table 20: Minimum Host System Requirements for GCP Deployment

Component	Value/Version	Notes
GCP Deployment	VM-based	
Instance Type	n2-standard-16	
CPU	Intel x86	The tested CPU is Intel Cascade Lake
Host OS	Rocky Linux 8.8 (Green Obsidian)	
Kernel Version	Rocky Linux 4.18.X	
NIC	VirtIO NIC	
Kubernetes (K8s)	1.25.x	The tested K8s version is 1.25.5. The K8s version for Google Cloud Marketplace Cloud-Native Router subscription is v1.27.5.
Calico	3.25.1	
Multus	4.0	

Table 20: Minimum Host System Requirements for GCP Deployment *(Continued)*

Component	Value/Version	Notes
Helm	3.9.x	
Container-RT	containerd 1.7.x	Other container runtimes may work but have not been tested with JCNR.
NOTE: The component versions listed in this table are expected to work with JCNR, but not every version or combination is tested in every release.		

Resource Requirements for GCP Deployment

Table 21 on page 210 lists the resource requirements for installing Cloud-Native Router on GCP.

Table 21: Resource Requirements for GCP Deployment

Resource	Value	Usage Notes
Data plane forwarding cores	1 core (1P + 1S)	
Service/Control Cores	0	

UIO Driver	VFIO-PCI	To enable, follow the steps below:
		`cat /etc/modules-load.d/vfio.conf`
		`vfio`
		`vfio-pci`
		Enable Unsafe IOMMU mode
		`echo Y > /sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts`
		`echo Y > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode`

Table 21: Resource Requirements for GCP Deployment *(Continued)*

Resource	Value	Usage Notes
Hugepages (1G)	6 Gi	See "Configure the Number of Huge Pages Available on a Node" on page 401.
Cloud-Native Router Controller cores	.5	
Cloud-Native Router vRouter Agent cores	.5	

Miscellaneous Requirements for GCP Deployment

Table 22 on page 211 lists additional requirements for deploying Cloud-Native Router on GCP.

Table 22: Miscellaneous Requirements for GCP Deployment

Requirement	Example

Set IOMMU and IOMMU-PT in GRUB.	Add the following line to **/etc/default/grub**.
	`GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
	Update grub and reboot.
	`grub2-mkconfig -o /boot/grub2/grub.cfg`
	`reboot`

Table 22: Miscellaneous Requirements for GCP Deployment *(Continued)*

Requirement	Example

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.
 	
NOTE: Applicable for L3 deployments only.
	Create a **/etc/modules-load.d/crpd.conf** file and add the following kernel modules to it:
	
	`tun`
	`fou`
	`fou6`
	`ipip`
	`ip_tunnel`
	`ip6_tunnel`
	`mpls_gso`
	`mpls_router`
	`mpls_iptunnel`
	`vrf`
	`vxlan`

Enable kernel-based forwarding on the Linux host.	`ip fou add port 6635 ipproto 137`

Table 22: Miscellaneous Requirements for GCP Deployment *(Continued)*

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td>Enable IP Forwarding for VMs in GCP.</td><td>Use one of these two methods to enable IP forwarding:
1. Specify it as an option while creating the VM. For example:
<code>gcloud compute instances create instance-name --can-ip-forward</code>
2. For an exisiting VM, enable IP forwarding by updating the compute instance via a file. For example:
<code>gcloud compute instances export transit-jcnr01 --project jcnr-ci-admin --zone us-west1-a --destination=instance_file_1</code>
Edit the instance file to set the value <code>canIpForward=true</code>.
Update the compute instance from the file:
<code>gcloud compute instances update-from-file transit-jcnr01 --project jcnr-ci-admin --zone us-west1-a --source=instance_file_1 --most-disruptive-allowed-action ALLOWED_ACTION</code></td></tr>
<tr><td>Enable Multi-IP subnet on Guest OS.</td><td><code>gcloud compute images create debian-9-multi-ip-subnet \</code>
<code>   --source-disk debian-9-disk \</code>
<code>   --source-disk-zone us-west1-a \</code>
<code>   --guest-os-features MULTI_IP_SUBNET</code></td></tr>
</table>

Table 22: Miscellaneous Requirements for GCP Deployment *(Continued)*

Requirement	Example
Add firewall rules for loopback address for VPC.	Configure the VPC firewall rule to allow ingress traffic with source filters set to the subnet range to which Cloud-Native Router is attached, along with the IP ranges or addresses for the loopback addresses. For example: Navigate to Firewall policies on the GCP console and create a firewall rule with the following attributes: 1. `Name`: Name of the firewall rule 2. `Network`: Choose the VPC network 3. `Priority`: 1000 4. `Direction`: Ingress 5. `Action on Match`: Allow 6. `Source filters`: 10.2.0.0/24, 10.51.2.0/24, 10.51.1.0/24, 10.12.2.2/32, 10.13.3.3/32 7. `Protocols`: all 8. `Enforcement`: Enabled where 10.2.0.0/24 is the subnet to which Cloud-Native Router is attached and 10.51.2.0/24, 10.51.1.0/24, 10.12.2.2/32, and 10.13.3.3/32 are loopback IP ranges.

Table 22: Miscellaneous Requirements for GCP Deployment *(Continued)*

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td>Exclude Cloud-Native Router interfaces from NetworkManager control.</td><td>
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 Kubernetes management and create problems.
To avoid NetworkManager from interfering with Cloud-Native Router interface configuration, exclude Cloud-Native Router interfaces from NetworkManager control. Here's an example on how to do this in some Linux distributions:

Create the /etc/NetworkManager/conf.d/crpd.conf file and list the interfaces that you don't want NetworkManager to manage.
For example:
<pre>[keyfile]
 unmanaged-devices+=interface-name:enp*;interface-
name:ens*</pre>
where <code>enp*</code> and <code>ens*</code> refer to your Cloud-Native Router interfaces.
NOTE: <code>enp*</code> indicates all interfaces starting with <code>enp</code>. For specific interface names, provided a comma-separated list.
Restart the NetworkManager service:
<pre>sudo systemctl restart NetworkManager</pre>
Edit the /etc/sysctl.conf file on the host and paste the following content in it:
<pre>net.ipv6.conf.default.addr_gen_mode=0
net.ipv6.conf.all.addr_gen_mode=0</pre>

</td></tr>
</table>

Table 22: Miscellaneous Requirements for GCP Deployment *(Continued)*

Requirement	Example

	`net.ipv6.conf.default.autoconf=0`
	`net.ipv6.conf.all.autoconf=0`
	
	4. Run the command `sysctl -p /etc/sysctl.conf` to load the new **sysctl.conf** values on the host.

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`

NOTE: Here are additional restrictions: • Cloud-Native Router supports only IPv4 for GCP. • Cloud-Native Router deployment on GCP supports only N8-standard for VM deployments. The N16-standard is not supported.	

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 23: 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 the Cloud-Native Router vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from Cloud-Native Router control plane
TCP	8075, 8076	Telemetry Information- Used for gNMI requests
TCP	9091	vRouter health check–cloud-native router checks to ensure the vRouter agent is running.
TCP	9092	vRouter health check–cloud-native router checks to ensure the vRouter DPDK is running.
TCP	50052	gRPC port–Cloud-Native Router listens on both IPv4 and IPv6
TCP	8081	Cloud-Native Router Deployer Port
TCP	24	cRPD SSH
TCP	830	cRPD NETCONF
TCP	666	**rpd**
TCP	1883	Mosquito mqtt–Publish/subscribe messaging utility
TCP	9500	**agentd** on cRPD

Table 23: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	21883	**na-mqttd**
TCP	50053	Default gNMI port that listens to the client subscription request
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

To deploy Cloud-Native Router on GCP, you can either download the Helm charts from the Juniper Networks software download site (see "Cloud-Native Router Software Download Packages" on page 387) or subscribe via the Google Cloud Marketplace.

NOTE: Before deploying Cloud-Native Router on GCP via Helm charts downloaded from the Juniper Networks software download site, you must whitelist the
https://enterprise.hub.juniper.net
URL as the Cloud-Native Router image repository.

Cloud-Native Router Licensing

See "Manage Cloud-Native Router Licenses" on page 352.

Customize Cloud-Native Router Helm Chart for GCP Deployment

IN THIS SECTION

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

Helm Chart Description for GCP Deployment

Customize the Helm chart using the **Juniper_Cloud_Native_Router_*<release>*/helmchart/jcnr/values.yaml** file. We provide a copy of the default **values.yaml** in "Cloud-Native Router Default Helm Chart" on page 389.

Table 24 on page 219 contains a description of the configurable attributes in **values.yaml** for a GCP deployment.

Table 24: Helm Chart Description for GCP Deployment

Key	Description
global	
installSyslog	Set to true to install syslog-ng.
registry	Defines the Docker registry for the Cloud-Native Router container images. The default value is `enterprise-hub.juniper.net`. The images provided in the tarball are tagged with the default registry name. If you choose to host the container images to a private registry, replace the default value with your registry URL.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
repository	(Optional) Defines the repository path for the Cloud-Native Router container images. This is a global key that takes precedence over the `repository` paths under the `common` section. Default is `jcnr-container-prod/`.
imagePullSecret	(Optional) Defines the Docker registry authentication credentials. You can configure credentials to either the Juniper Networks enterprise-hub.juniper.net registry or your private registry.
registryCredentials	Base64 representation of your Docker registry credentials. See "Configure Repository Credentials" on page 398 for more information.
secretName	Name of the secret object that will be created.
common	Defines repository paths and tags for the Cloud-Native Router container images. Use defaults unless using a private registry.
repository	Defines the repository path. The default value is `jcnr-container-prod/`. 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.
readinessCheck	Set to true to enable Cloud-Native Router Readiness preflight and postflight checks during installation. Comment this out or set to false to disable Cloud-Native Router Readiness preflight and postflight checks. Preflight checks verify that your infrastructure can support JCNR. Preflight checks take place before Cloud-Native Router is installed. Postflight checks verify that your Cloud-Native Router installation is working properly. Postflight checks take place after Cloud-Native Router is installed. See "Cloud-Native Router Readiness Checks" on page 362.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
replicas	(Optional) Indicates the number of replicas for cRPD. Default is 1. The value for this key must be specified for multi-node clusters. The value is equal to the number of nodes running JCNR.
noLocalSwitching	Not applicable.
iamRole	Not applicable.

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: Use the L3 only section to configure fabric interfaces for GCP. The L2 only and L2-L3 sections are not applicable for GCP deployments. Do not configure `interface_mode` for any of the interfaces.
	
	For example:
	
	`# L3  only`
	`- eth1:`
	`    ddp: "off"`
	`- eth2:`
	`    ddp: "off"`
	
	See "Cloud-Native Router Interfaces Overview" on page 14 for more information.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description

subnet	An alternative mode of input to 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 in a multi-node cluster.

ddp	Not applicable.
interface_mode	Not applicable.
vlan-id-list	Not applicable.
storm-control-profile	Not applicable.
native-vlan-id	Not applicable.
no-local-switching	Not applicable.
qosSchedulerProfileName	Specifies the QoS scheduler profile applicable to this interface. See the `qosSchedulerProfiles` section. If you don't specify a profile, then the QoS scheduler is disabled for this interface, which means that packets are scheduled with no regard to traffic class.
fabricWorkloadInterface	Not applicable.
log_level	Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave it set to the default INFO unless instructed to change it by Juniper Networks support.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
log_path	The defined directory stores various JCNR-related descriptive logs such as **contrail-vrouter-agent.log**, **contrail-vrouter-dpdk.log**, etc. Default is **/var/log/jcnr/**.
syslog_notifications	Indicates the absolute path to the file that stores `syslog-ng` generated notifications in JSON format. Default is **/var/log/jcnr/jcnr_notifications.json**.
corePattern	Indicates the `core_pattern` for the core file. If left blank, then Cloud-Native Router pods will not overwrite the default pattern on the host. **NOTE**: Set the `core_pattern` on the host before deploying JCNR. You can 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 to the `core` file. Default is **/var/crash**.

nodeAffinity	(Optional) Defines labels on nodes to determine where to place the vRouter pods.
	
	By default the vRouter pods are deployed to all nodes of a cluster.
	
	In the example below, the node affinity label is defined as `key1=jcnr`. You must apply this label to each node where Cloud-Native Router is to be deployed:
	
	`nodeAffinity:`
	`- key: key1`
	`operator: In`
	`values:`
	`- jcnr`
	
	NOTE: This key is a global setting.

key	Key-value pair that represents a node label that must be matched to apply the node affinity.

Table 24: Helm Chart Description for GCP Deployment ***(Continued)***

Key	Description
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 path with the path in your distribution (for example, **/var/opt/cni/bin**).
grpcTelemetryPort	(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50053.
grpcVrouterPort	(Optional) Default is 50052. Configure to override.
vRouterDeployerPort	(Optional) Default is 8081. Configure to override.
jcnr-vrouter	
cpu_core_mask	If present, this indicates that you want to use static CPU allocation to allocate cores to the forwarding plane. This value should be a comma-delimited list of isolated CPU cores that you want to statically allocate to the forwarding plane (for example, `cpu_core_mask: "2,3,22,23"`). Use the cores not used by the host OS. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Cloud-Native Router Readiness preflight checks, if enabled, will fail the installation if you specify both.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
guaranteedVrouterCpus	If present, this indicates that you want to use the Kubernetes CPU Manager to allocate CPU cores to the forwarding plane. This value should be the number of guaranteed CPU cores that you want the Kubernetes CPU Manager to allocate to the forwarding plane. You should set this value to at least one more than the number of forwarding cores. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Using both can lead to unpredictable behavior.
dpdkCtrlThreadMask	Specifies the CPU core(s) to allocate to vRouter DPDK control threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `serviceCoreMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.
serviceCoreMask	Specifies the CPU core(s) to allocate to vRouter DPDK service threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `dpdkCtrlThreadMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
numServiceCtrlThreadCPU	Specifies the number of CPU cores to allocate to vRouter DPDK service/control traffic when using the Kubernetes CPU Manager. This number should be smaller than the number of `guaranteedVrouterCpus` cores. The remaining `guaranteedVrouterCpus` cores are allocated for forwarding. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
numberOfSchedulerLcores	The number of CPU cores that you want Kubernetes CPU Manager to dedicate to your QoS schedulers. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
restoreInterfaces	Set to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts or if Cloud-Native Router is uninstalled.
bondInterfaceConfigs	Not applicable.
mtu	Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default is 9000.
qosSchedulerProfiles	Defines the QoS scheduler profiles that are referenced from the `fabricInterface` section.
sched_profile_1	The name of the QoS scheduler profile.
	cpu Specify the CPU core(s) to dedicate to the scheduler. If `cpu_core_mask` is specified, this should be a unique additional core(s). **bandwidth** Specify the bandwidth in Gbps.
stormControlProfiles	Not applicable.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
dpdkCommandAdditionalArgs	Pass any additional DPDK command line parameters. The `--yield_option 0` is set by default and implies the DPDK forwarding cores will not yield their assigned CPU cores. Other common parameters that can be added are `tx` and `rx` descriptors and `mempool`. For example: `dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 2048 --dpdk_rxd_sz 2048 --vr_mempool_sz 131072"` **NOTE**: Changing the number of `tx` and `rx` descriptors and the `mempool` size affects the number of huge pages required. If you make explicit changes to these parameters, set the number of huge pages to 10 (x 1 GB). See "Configure Huge Pages" on page 401 for information on how to configure huge pages.
dpdk_monitoring_thread_config	(Optional) Enables a monitoring thread for the vRouter DPDK container. Every `loggingInterval` seconds, a log containing the information indicated by `loggingMask` is generated.
loggingMask	Specifies the information to be generated. Represented by a bitmask with bit positions as follows: • 0b001 is the nl_counter • 0b010 is the lcore_timestamp • 0b100 is the profile_histogram
loggingInterval	Specifies the log generation interval in seconds.
ddp	Not applicable.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
twampPort	(Optional) The TWAMP session reflector port (if you want TWAMP sessions to use vRouter timestamps). The vRouter listens to TWAMP test messages on this port and inserts/ overwrites timestamps in TWAMP test messages. Timestamping TWAMP messages at the vRouter (instead of at cRPD) leads to more accurate measurements. Valid values are 862 and 49152 through 65535. If this parameter is absent, then the vRouter does not insert or overwrite timestamps in the TWAMP session. Timestamps are taken and inserted by cRPD instead. See *Two-Way Active Measurement Protocol (TWAMP)*.
vrouter_dpdk_uio_driver	The uio driver is `vfio-pci`.
agentModeType	Set to dpdk.
fabricRpfCheckDisable	Set to false to enable the RPF check on all Cloud-Native Router fabric interfaces. By default, RPF check is disabled.
telemetry	(Optional) Configures cRPD telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
disable	Set to true to disable cRPD telemetry. Default is false, which means that cRPD telemetry is enabled by default.
metricsPort	The port that the cRPD telemetry exporter is listening to Prometheus queries on. Default is 8072.
logLevel	One of warn, warning, info, debug, trace, or verbose. Default is info.
gnmi	(Optional) Configures cRPD gNMI settings.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
	enable Set to true to enable the cRPD telemetry exporter to respond to gNMI requests.
vrouter	
telemetry	(Optional) Configures vRouter telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
	metricsPort Specify the port that the vRouter telemetry exporter listens to Prometheus queries on. Default is 8070.
	logLevel One of warn, warning, info, debug, trace, or verbose. Default is info.
	gnmi (Optional) Configures vRouter gNMI settings. enable - Set to true to enable the vRouter telemetry exporter to respond to gNMI requests.
persistConfig	Set to true if you want Cloud-Native Router pod configuration to persist even after uninstallation. This option can only be set for L2 mode deployments. Default is false.
enableLocalPersistence	Set to true if you're using the cRPD CLI or NETCONF to configure JCNR. When set to true, the cRPD CLI and NETCONF configuration persists through node reboots, cRPD pod restarts, and Cloud-Native Router upgrades. Default is false.
interfaceBoundType	Not applicable.
networkDetails	Not applicable.
networkResources	Not applicable.

Table 24: Helm Chart Description for GCP Deployment *(Continued)*

Key	Description
contrail-tools	
install	Set to true to install contrail-tools (used for debugging).

Customize Cloud-Native Router Configuration

SUMMARY

Read this topic to understand how to customize Cloud-Native Router configuration using a Configlet custom resource.

IN THIS SECTION

Configlet Custom Resource

Starting with Juniper Cloud-Native Router (JCNR) Release 24.2, we support customizing Cloud-Native Router configuration using a configlet custom resource. The configlet can be generated either by rendering a predefined template of supported Junos configuration or using raw configuration. The generated configuration is validated and deployed on the Cloud-Native Router controller (cRPD) as one or more Junos configuration groups.

> **NOTE**: You can configure Cloud-Native Router using either configlets or the cRPD CLI or NETCONF. If you use the cRPD CLI or NETCONF, be sure to enable local persistence in **values.yaml** (`enableLocalPersistence: true`) so that your CLI or NETCONF configuration persists across reboots and upgrades.

NOTE: Using both configlets and the cRPD CLI or NETCONF to configure Cloud-Native Router may lead to unpredictable behavior. Use one or the other, but not both.

Configuration Examples

You create a configlet custom resource of the kind `Configlet` in the `jcnr` namespace. You provide raw configuration as Junos `set` commands.

Use `crpdSelector` to control where the configlet applies. The generated configuration is deployed to cRPD pods on nodes matching the specified label only. If `crpdSelector` is not defined, the configuration is applied to all cRPD pods in the cluster.

An example configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample             # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker                   # <-- Node label to select the cRPD pods
```

You can also use a templatized configlet yaml that contains keys or variables. The values for variables are provided by a configletDataValue custom resource, referenced by `configletDataValueRef` . An example templatized configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-with-template      # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
```

```
  crpdSelector:
    matchLabels:
      node: worker                         # <-- Node label to select the cRPD pods
  configletDataValueRef:
    name: "configletdatavalue-sample"       # <-- Configlet Data Value resource name
```

To render configuration using the template, you must provide key:value pairs in the ConfigletDataValue custom resource:

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"           # <-- Key:Value pair
  }
```

The generated configuration is validated and applied to all or selected cRPD pods as a Junos Configuration Group.

Applying the Configlet Resource

The configlet resource can be used to apply configuration to selected or all cRPD pods either when Cloud-Native Router is deployed or once the cRPD pods are up and running. Let us look at configlet deployment in detail.

Applying raw configuration

1. Create raw configuration configlet yaml. The example below configures a loopback interface in cRPD.

   ```
   cat configlet-sample.yaml
   ```

   ```
   apiVersion: configplane.juniper.net/v1
   kind: Configlet
   metadata:
     name: configlet-sample
   ```

```
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker
```

2. Apply the configuration using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                              AGE
configlet-sample-node1            10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

```
kubectl describe nodeconfiglet configlet-sample-node1 -n jcnr
```

   The following output has been trimmed for brevity:

```
Name:         configlet-sample-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
Annotations:  <none>
```

```
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 10.10.10.1/32;
            }
        }
    }
}
```

NOTE: The configuration generated using configlets is applied to cRPD as configuration groups. We therefore recommend that you not use configuration groups when specifying your configlet.

Applying templatized configuration

1. Create the templatized configlet yaml and the configlet data value yaml for key:value pairs.

```
cat configlet-sample-template.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-template
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: master
  configletDataValueRef:
    name: "configletdatavalue-sample"
```

```
cat configletdatavalue-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"
  }
```

2. Apply the configuration using the `kubectl apply` command, starting with the config data value yaml.

```
kubectl apply -f configletdatavalue-sample.yaml
```

```
configletdatavalue.configplane.juniper.net/configletdatavalue-sample created
```

```
kubectl apply -f configlet-sample-template.yaml
```

```
configlet.configplane.juniper.net/configlet-sample-template created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                                        AGE
configlet-sample-template-node1             10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

```
kubectl describe nodeconfiglet configlet-sample-template-node1 -n jcnr
```

   The following output has been trimmed for brevity:

```
Name:          configlet-sample-template-node1
Namespace:     jcnr
Labels:        core.juniper.net/nodeName=node1
Annotations:   <none>
API Version:   configplane.juniper.net/v1
Kind:          NodeConfiglet
```

```
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample-template
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample-template
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32;
            }
        }
    }
}
```

Modifying the Configlet

You can modify a configlet resource by changing the yaml file and reapplying it using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample configured
```

Any changes to existing configlet resource are reconciled by replacing the configuration group on cRPD.

You can delete the configuration group by deleting the configlet resource using the `kubectl delete` command.

```
kubectl delete configlet configlet-sample -n jcnr
```

```
configlet.configplane.juniper.net "configlet-sample" deleted
```

Troubleshooting

If you run into problems, check the `contrail-k8s-deployer` logs. For example:

```
kubectl logs contrail-k8s-deployer-8ff895cc5-cbfwm -n contrail-deploy
```

6

CHAPTER

Install Cloud-Native Router on Wind River Cloud Platform

IN THIS CHAPTER

Install and Verify Juniper Cloud-Native Router for Wind River Deployment

SUMMARY

The Juniper Cloud-Native Router uses 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

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 "System Requirements for Wind River Deployment" on page 248 section to ensure the server has all the required configuration.
2. Download the desired Cloud-Native Router software package to the directory of your choice.

   You have the option of downloading the package to install Cloud-Native Router only or downloading the package to install JNCR together with Juniper cSRX. See "Cloud-Native Router Software Download Packages" on page 387 for a description of the packages available. If you don't want to install Juniper cSRX now, you can always choose to install Juniper cSRX on your working Cloud-Native Router installation later.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

   ```
   tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
   ```
4. Change directory to the main installation directory.
   - If you're installing Cloud-Native Router only, then:

     ```
     cd Juniper_Cloud_Native_Router_<release>
     ```

This directory contains the Helm chart for Cloud-Native Router only.

- If you're installing Cloud-Native Router and cSRX at the same time, then:

```
cd Juniper_Cloud_Native_Router_CSRX_<release>
```

This directory contains the combination Helm chart for Cloud-Native Router and cSRX.

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

5. View the contents in the current directory.

```
ls
helmchart  images  README.md  secrets
```

6. Change to the **helmchart** directory and expand the Helm chart.

```
cd helmchart
```

- For Cloud-Native Router only:

```
ls
jcnr-<release>.tgz
```

```
tar -xzvf jcnr-<release>.tgz
```

```
ls
jcnr   jcnr-<release>.tgz
```

The Helm chart is located in the **jcnr** directory.

- For the combined Cloud-Native Router and cSRX:

```
ls
jcnr_csrx-<release>.tgz
```

```
tar -xzvf jcnr_csrx-<release>.tgz
```

```
ls
jcnr_csrx   jcnr_csrx-<release>.tgz
```

The Helm chart is located in the **jcnr_csrx** directory.

7. 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 398 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 399 for instructions on how to import images to the local container runtime.

8. Follow the steps in "Installing Your License" on page 353 to install your Cloud-Native Router license.

9. 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**.

10. Apply **secrets/jcnr-secrets.yaml** to the cluster.

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

11. If desired, configure how cores are assigned to the vRouter DPDK containers. See "Allocate CPUs to the Cloud-Native Router Forwarding Plane" on page 355.

12. Customize the Helm chart for your deployment using the **helmchart/jcnr/values.yaml** or **helmchart/jcnr_csrx/values.yaml** file.

    See "Customize Cloud-Native Router Helm Chart for Wind River Deployment" on page 261 for descriptions of the Helm chart configurations.

13. Optionally, customize Cloud-Native Router configuration.

    See, "Customize Cloud-Native Router Configuration " on page 62 for creating and applying the cRPD customizations.

14. If you're installing Juniper cSRX now, then follow the procedure in "Apply the cSRX License and Configure cSRX" on page 338.

15. Label the nodes where you want Cloud-Native Router to be installed based on the `nodeaffinity` configuration (if defined in the `values.yaml`). For example:

```
kubectl label nodes ip-10.0.100.17.lab.net key1=jcnr --overwrite
```

16. Deploy the Juniper Cloud-Native Router using the Helm chart.

    Navigate to the **helmchart/jcnr** or the **helmchart/jcnr_csrx** directory and run the following command:

```
helm install jcnr .
```

or

```
helm install jcnr-csrx .
```

```
NAME: jcnr
LAST DEPLOYED: Fri Dec 22 06:04:33 2023
NAMESPACE: default
STATUS: deployed
```

```
REVISION: 1
TEST SUITE: None
```

17. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

Sample output:

```
NAME        NAMESPACE    REVISION        UPDATED
STATUS         CHART                       APP VERSION
jcnr        default      1               2023-12-22 06:04:33.144611017 -0400 EDT
deployed       jcnr-<version>                  <version>
```

Verify Installation

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

NOTE: The output shown in this example procedure is affected by the number of nodes in the cluster. The output you see in your setup may differ in that regard.

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          jcnr-0-dp-contrail-vrouter-nodes-b2jxp         2/2     Running   0               87s
contrail          jcnr-0-dp-contrail-vrouter-nodes-vrdpdk-g7wrk  1/1     Running   0               87s
jcnr              jcnr-0-crpd-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      jcnr-0-dp-contrail-vrouter-nodes         1         1         1       1            1
<none>                   90m
contrail      jcnr-0-dp-contrail-vrouter-nodes-vrdpdk  1         1         1       1            1
<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        jcnr-0-crpd   1/1     27m
```

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

   a. View the *Access cRPD CLI* 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 *Access vRouter CLI* 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.

6. If desired, download and run Validation Factory topology tests. See "Validation Factory" on page 364.

System Requirements for Wind River Deployment

IN THIS SECTION

Read this section to understand the system, resource, port, and licensing requirements for installing Juniper Cloud-Native Router on a Wind River deployment. We provide requirements for both pre-bound and non-pre-bound SR-IOV interfaces.

Minimum Host System Requirements on a Wind River Deployment

Table 25 on page 249 lists the host system requirements for installing Cloud-Native Router on a Wind River deployment.

Table 25: Cloud-Native Router Minimum Host System Requirements on a Wind River Deployment

Component	Value/Version	Notes
CPU	Intel x86	The tested CPU is Intel(R) Xeon(R) Silver 4314 CPU @ 2.40GHz
Host OS	Debian GNU/Linux (depends on Wind River Cloud Platform version)	
Kernel Version	5.10.x	5.10.0-6-amd64

Table 25: Cloud-Native Router Minimum Host System Requirements on a Wind River Deployment *(Continued)*

Component	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 • Mellanox ConnectX-6 • Mellanox ConnectX-7	Support for Mellanox NICs is considered a Juniper Technology Preview ("Tech Preview" on page 452) feature. When using Mellanox NICs, ensure your interface names do not exceed 11 characters in length.
Wind River Cloud Platform	22.12	
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 1.24	The tested K8s version is 1.24.4
Calico	Version 3.24.x	
Multus	Version 3.8	

Table 25: Cloud-Native Router Minimum Host System Requirements on a Wind River Deployment *(Continued)*

Component	Value/Version	Notes
Helm	3.9.x	
Container-RT	• containerd 1.4.x • crictl 1.21.x	Other container runtimes may work but have not been tested with JCNR.
NOTE: The component versions listed in this table are expected to work with JCNR, but not every version or combination is tested in every release.		

Resource Requirements on a Wind River Deployment

Table 26 on page 251 lists the resource requirements for installing Cloud-Native Router on a Wind River deployment.

Table 26: Resource Requirements on a Wind River Deployment

Resource	Value	Usage Notes
Data plane forwarding cores	1 core (1P + 1S)	
Service/Control Cores	0	

Table 26: Resource Requirements on a Wind River Deployment *(Continued)*

Resource	Value	Usage Notes

Hugepages (1G)	6 Gi	Lock the controller and get the memory processors using below command:
		`source /etc/platform/openrc`
		`system host-lock controller-0`
		`system host-memory-list controller-0`
		To set the huge pages, run the following command for each controller:
		`system host-memory-modify controller-0 0 -1G 6`
		`system host-memory-modify controller-0 1 -1G 6`
		View the huge pages with the following command:
		`system host-memory-list controller-0`
		Unlock the controller:
		`system host-unlock controller-0`

Cloud-Native Router Controller cores	.5	
Cloud-Native Router vRouter Agent cores	.5	

Miscellaneous Requirements on a Wind River Deployment

Table 27 on page 253 lists the additional requirements for installing Cloud-Native Router on a Wind River deployment.

Table 27: Miscellaneous Requirements on a Wind River Deployment

Requirement	Example
Enable the host with SR-IOV and VT-d in the system's BIOS.	Depends on BIOS.

Isolate CPUs from the kernel scheduler.	`source /etc/platform/openrc`
	`system host-lock controller-0`
	`system host-cpu-list controller-0`
	`system host-cpu-modify -f application-isolated -c 4-59 controller-0`
	`system host-unlock controller-0`

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.
 	
NOTE: Applicable for L3 deployments only.
	Create a conf file and add the kernel modules:
	
	`cat /etc/modules-load.d/crpd.conf`
	`tun`
	`fou`
	`fou6`
	`ipip`
	`ip_tunnel`
	`ip6_tunnel`
	`mpls_gso`
	`mpls_router`
	`mpls_iptunnel`
	`vrf`
	`vxlan`

Enable kernel-based forwarding on the Linux host.	`ip fou add port 6635 ipproto 137`

Table 27: Miscellaneous Requirements on a Wind River Deployment *(Continued)*

Requirement	Example
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`

Requirements for Pre-Bound SR-IOV Interfaces on a Wind River Deployment

In a Wind River deployment, you typically bind all your Cloud-Native Router interfaces to the vfio DPDK driver before you deploy JCNR. Table 28 on page 255 shows an example of how you can do this on an SR-IOV-enabled interface on a host.

NOTE: We support pre-binding interfaces for Cloud-Native Router L3 mode deployments only.

Table 28: Requirements for Pre-Bound SR-IOV Interfaces on a Wind River Deployment

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td>Pre-bind the Cloud-Native Router interfaces to the vfio DPDK driver.</td><td>

```
source /etc/platform/openrc

system host-lock controller-0
system host-label-assign controller-0 sriovdp=enabled   # <-- Label node to accept SR-IOV-enabled
                                                        #     deployments.

system host-label-assign controller-0 kube-cpu-mgr-policy=static
system host-label-assign controller-0 kube-topology-mgr-policy=restricted   # <-- see note below

system datanetwork-add datanet0 flat   # <-- Create datanet0 network. You'll define this in a NAD
                                       #     later.

DTNIF=enp175s0f0
system host-if-modify -m 1500 -n $DTNIF -c pci-sriov -N 8 controller-0 $DTNIF --vf-driver=netdevice
                                                      # ^ Enable 8 (for example) VFs on enp175s0f0.

system host-if-add -c pci-sriov controller-0 srif0 vf $DTNIF -N 1 --vf-driver=vfio
                                             # ^ Create srif0 interface that uses one of the VFs
                                             #   and bind to vfio driver.

IFUUID=$(system host-if-list 1 | awk '{if ($4 == "srif0") {print $2}}')
system interface-datanetwork-assign 1 $IFUUID datanet0  #  <-- Attach srif0 interface to datanet0
network.

system host-unlock 1
```

NOTE: On hosts with a single NUMA node or where all NICs are attached to the same NUMA node, set `kube-topology-mgr-policy=restricted`.
On hosts with multiple NUMA nodes where the NICs are spread across NUMA nodes, set `kube-topology-mgr-policy=best-effort`.</td></tr>
</table>

Table 28: Requirements for Pre-Bound SR-IOV Interfaces on a Wind River Deployment *(Continued)*

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td>Create and apply the Network Attachment Definition that attaches the datanet0 network defined above.</td><td>Create a yaml file for the Network Attachment Definition. For example:
<pre>
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: srif0net0
  annotations:
    k8s.v1.cni.cncf.io/resourceName: intel.com/pci_sriov_net_datanet0
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "sriov",
      "spoofchk": "off",
      "trust": "on"
  }'
</pre>
Apply the yaml to attach the datanet0 network:
<pre>
kubectl apply -f srif0net0.yaml
</pre>
where srif0net0.yaml is the file that contains the Network Attachment Definition above.</td></tr>
</table>

Table 28: Requirements for Pre-Bound SR-IOV Interfaces on a Wind River Deployment *(Continued)*

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td>Update the Helm chart values.yaml to use the defined networks.</td><td>Here's an example of using two networks, datanet0/srif0net0 and datanet1/srif1net1.
<pre>
jcnr-vrouter:
  guaranteedVrouterCpus: 4
  interfaceBoundType: 1

  networkDetails:
  - ddp: "off"
    name: srif0net0
    namespace: default
  - ddp: "off"
    name: srif1net1
    namespace: default

  networkResources:
    limits:
      intel.com/pci_sriov_net_datanet0: "1"
      intel.com/pci_sriov_net_datanet1: "1"
    requests:
      intel.com/pci_sriov_net_datanet0: "1"
      intel.com/pci_sriov_net_datanet1: "1"
</pre>
Here's an example of using a bond interface attached to two networks (datanet0/srif0net0 and datanet1/srif1net1) and a regular interface attached to a third network (datanet2/srif2net2).
<pre>
jcnr-vrouter:
  guaranteedVrouterCpus: 4
  interfaceBoundType: 1

  bondInterfaceConfigs:
  - mode: 1
    name: bond0
    slaveNetworkDetails:
    - name: srif0net0
      namespace: default
    - name: srif1net1
      namespace: default

  networkDetails:
</pre></td></tr>
</table>

Table 28: Requirements for Pre-Bound SR-IOV Interfaces on a Wind River Deployment *(Continued)*

Requirement	Example

	<pre>- ddp: "off"
	  name: bond0
	- ddp: "off"
	  name: srif2net2
	  namespace: default
	
	networkResources:
	  limits:
	    intel.com/pci_sriov_net_datanet0: "1"
	    intel.com/pci_sriov_net_datanet1: "1"
	    intel.com/pci_sriov_net_datanet2: "1"
	  requests:
	    intel.com/pci_sriov_net_datanet0: "1"
	    intel.com/pci_sriov_net_datanet1: "1"
	    intel.com/pci_sriov_net_datanet2: "1"</pre>

Requirements for Non-Pre-Bound SR-IOV Interfaces on a Wind River Deployment

In some situations, you might want to run with non-pre-bound interfaces. Table 29 on page 259 shows the requirements for non-pre-bound interfaces.

Table 29: Requirements for Non-Pre-Bound SR-IOV Interfaces on a Wind River Deployment

Requirement	Example

Configure IPv4 and IPv6 addresses for the non-pre-bound interfaces allocated to JCNR.	`source /etc/platform/openrc`
	`system host-lock controller-0`
	`system host-if-modify -n ens1f0 -c platform --ipv4-mode static controller-0 ens1f0`
	`system host-addr-add 1 ens1f0 11.11.11.29 24`
	`system host-if-modify -n ens1f0 -c platform --ipv6-mode static controller-0 ens1f0`
	`system host-addr-add 1 ens1f0 abcd::11.11.11.29 112`
	`system host-if-list controller-0`
	`system host-addr-list controller-0`
	`system host-unlock controller-0`

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 30: 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 the Cloud-Native Router vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from Cloud-Native Router control plane
TCP	8075, 8076	Telemetry Information- Used for gNMI requests

Table 30: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	9091	vRouter health check–cloud-native router checks to ensure the vRouter agent is running.
TCP	9092	vRouter health check–cloud-native router checks to ensure the vRouter DPDK is running.
TCP	50052	gRPC port–Cloud-Native Router listens on both IPv4 and IPv6
TCP	8081	Cloud-Native Router Deployer Port
TCP	24	cRPD SSH
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	50053	Default gNMI port that listens to the client subscription request
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

See "Cloud-Native Router Software Download Packages" on page 387.

Cloud-Native Router Licensing

See "Manage Cloud-Native Router Licenses" on page 352.

Customize Cloud-Native Router Helm Chart for Wind River Deployment

IN THIS SECTION

Read this topic to learn about the deployment configuration available for the Juniper Cloud-Native Router on a Wind River Deployment.

You can deploy and operate Juniper Cloud-Native Router in the L3 mode on a Wind River deployment. You configure the deployment mode by editing the appropriate attributes in the `values.yaml` file prior to deployment.

Helm Chart Description for Wind River Deployment

Customize the Helm chart using the **Juniper_Cloud_Native_Router_*<release>*/helmchart/jcnr/values.yaml** file. We provide a copy of the default **values.yaml** in "Cloud-Native Router Default Helm Chart" on page 389.

Table 31 on page 261 contains a description of the configurable attributes in **values.yaml** for a Wind River deployment.

Table 31: Helm Chart Description for Wind River Deployment

Key	Description
global	

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
installSyslog	Set to true to install syslog-ng.
registry	Defines the Docker registry for the Cloud-Native Router container images. The default value is `enterprise-hub.juniper.net`. The images provided in the tarball are tagged with the default registry name. If you choose to host the container images to a private registry, replace the default value with your registry URL.
repository	(Optional) Defines the repository path for the Cloud-Native Router container images. This is a global key that takes precedence over the `repository` paths under the `common` section. Default is `jcnr-container-prod/`.
imagePullSecret	(Optional) Defines the Docker registry authentication credentials. You can configure credentials to either the Juniper Networks enterprise-hub.juniper.net registry or your private registry.
registryCredentials	Base64 representation of your Docker registry credentials. See "Configure Repository Credentials" on page 398 for more information.
secretName	Name of the secret object that will be created.
common	Defines repository paths and tags for the various Cloud-Native Router container images. Use default unless using a private registry.
repository	Defines the repository path. The default value is `jcnr-container-prod/`. 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.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
readinessCheck	Set to true to enable Cloud-Native Router Readiness preflight and postflight checks during installation. Comment this out or set to false to disable Cloud-Native Router Readiness preflight and postflight checks. Preflight checks verify that your infrastructure can support JCNR. Preflight checks take place before Cloud-Native Router is installed. Postflight checks verify that your Cloud-Native Router installation is working properly. Postflight checks take place after Cloud-Native Router is installed. See "Cloud-Native Router Readiness Checks" on page 362.
replicas	(Optional) Indicates the number of replicas for cRPD. Default is 1. The value for this key must be specified for multi-node clusters. The value is equal to the number of nodes running JCNR.
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 to L2 and L2-L3 deployments. Enabling this key provides the functionality on all access interfaces. To enable the functionality on trunk interfaces, configure `no-local-switching` in `fabricInterface`. See *Prevent Local Switching* for more details.
iamRole	Not applicable.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description

fabricInterface	Provide a list of interfaces to be bound to 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.
	
	For example:
	
	`# L3 only`
	`- eth1:`
	`    ddp: "off"`
	
	This attribute and all of its child attributes are only applicable when running with non-pre-bound SR-IOV interfaces.
	
	Comment out these attributes when running with pre-bound SR-IOV interfaces.

subnet	An alternative mode of input to 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 in a multi-node cluster.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
ddp	(Optional) Indicates the interface-level Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at the NIC for traffic like GTPU, SCTP, etc. See *Enabling Dynamic Device Personalization (DDP) on Individual Interfaces* for more details. Options include auto, on, or off. Default is off. **NOTE**: The interface level `ddp` takes precedence over the global `ddp` configuration.
interface_mode	Not applicable.
vlan-id-list	Not applicable.
storm-control-profile	Not applicable.
native-vlan-id	Not applicable.
no-local-switching	Not applicable.
qosSchedulerProfileName	Specifies the QoS scheduler profile applicable to this interface. See the `qosSchedulerProfiles` section. If you don't specify a profile, then the QoS scheduler is disabled for this interface, which means that packets are scheduled with no regard to traffic class.
fabricWorkloadInterface	Not applicable.
log_level	Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave it set to the default INFO unless instructed to change it by Juniper Networks support.
log_path	The defined directory stores various JCNR-related descriptive logs such as **contrail-vrouter-agent.log**, **contrail-vrouter-dpdk.log**, etc. Default is **/var/log/jcnr/**.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
syslog_notifications	Indicates the absolute path to the file that stores `syslog-ng` generated notifications in JSON format. Default is **/var/log/jcnr/jcnr_notifications.json**.
corePattern	Indicates the `core_pattern` for the core file. If left blank, then Cloud-Native Router pods will not overwrite the default pattern on the host. **NOTE**: Set the `core_pattern` on the host before deploying JCNR. You can 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 to the `core` file. Default is **/var/crash**.

nodeAffinity	(Optional) Defines labels on nodes to determine where to place the vRouter pods.
	
	By default the vRouter pods are deployed to all nodes of a cluster.
	
	In the example below, the node affinity label is defined as `key1=jcnr`. You must apply this label to each node where Cloud-Native Router is to be deployed:
	
	`nodeAffinity:`
	`- key: key1`
	`operator: In`
	`values:`
	`- jcnr`
	
	NOTE: This key is a global setting.

key	Key-value pair that represents a node label that must be matched to apply the node affinity.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
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	Set to **/var/opt/cni/bin**.
grpcTelemetryPort	(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50053.
grpcVrouterPort	(Optional) Default is 50052. Configure to override.
vRouterDeployerPort	(Optional) Default is 8081. Configure to override.
jcnr-vrouter	
cpu_core_mask	If present, this indicates that you want to use static CPU allocation to allocate cores to the forwarding plane. This value should be a comma-delimited list of isolated CPU cores that you want to statically allocate to the forwarding plane (for example, `cpu_core_mask: "2,3,22,23"`). Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Cloud-Native Router Readiness preflight checks, if enabled, will fail the installation if you specify both.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
guaranteedVrouterCpus	If present, this indicates that you want to use the Kubernetes CPU Manager to allocate CPU cores to the forwarding plane. This value should be the number of guaranteed CPU cores that you want the Kubernetes CPU Manager to allocate to the forwarding plane. You should set this value to at least one more than the number of forwarding cores. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Using both can lead to unpredictable behavior.
dpdkCtrlThreadMask	Specifies the CPU core(s) to allocate to vRouter DPDK control threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `serviceCoreMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.
serviceCoreMask	Specifies the CPU core(s) to allocate to vRouter DPDK service threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `dpdkCtrlThreadMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
numServiceCtrlThreadCPU	Specifies the number of CPU cores to allocate to vRouter DPDK service/control traffic when using the Kubernetes CPU Manager. This number should be smaller than the number of `guaranteedVrouterCpus` cores. The remaining `guaranteedVrouterCpus` cores are allocated for forwarding. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
numberOfSchedulerLcores	The number of CPU cores that you want Kubernetes CPU Manager to dedicate to your QoS schedulers. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
restoreInterfaces	Set to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts or if Cloud-Native Router is uninstalled.
bondInterfaceConfigs	(Optional) Enable bond interface configurations for L3 mode deployments. **NOTE**: The `bondInterfaceConfigs` attribute and its child attributes are only applicable when running with pre-bound SR-IOV interfaces. Comment out these attributes when running with non-pre-bound SR-IOV interfaces.
name	Name of the bond interface.
mode	Set to 1 (active-backup).
slaveInterfaces	Not applicable.
primaryInterface	Not applicable.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
slaveNetworkDetails	Information on the slave network interfaces (in name / namespace pairs) when using Network Attachment Definitions for L3 mode deployments. For an example on how to use this, see "Requirements for Pre-Bound SR-IOV Interfaces on a Wind River Deployment" on page 254.
	name Name of the slave interface.
	namespace Namespace for the slave interface.
mtu	Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default value is 9000.
qosSchedulerProfiles	Defines the QoS scheduler profiles that are referenced from the `fabricInterface` section.
sched_profile_1	The name of the QoS scheduler profile.
	cpu Specify the CPU core(s) to dedicate to the scheduler. If `cpu_core_mask` is specified, this should be a unique additional core(s). **bandwidth** Specify the bandwidth in Gbps.
stormControlProfiles	Configure the rate limit profiles for BUM traffic on fabric interfaces in bytes per second. See */Content/l2-bum-rate-limiting_xi931744_1_1.dita* for more details.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
dpdkCommandAdditionalArgs	Pass any additional DPDK command line parameters. The `--yield_option 0` is set by default and implies the DPDK forwarding cores will not yield their assigned CPU cores. Other common parameters that can be added are `tx` and `rx` descriptors and `mempool`. For example: `dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 2048 --dpdk_rxd_sz 2048 --vr_mempool_sz 131072"` **NOTE**: Changing the number of `tx` and `rx` descriptors and the `mempool` size affects the number of huge pages required. If you make explicit changes to these parameters, set the number of huge pages to 10 (x 1 GB). See "Resource Requirements on a Wind River Deployment" on page 251 in "System Requirements for Wind River Deployment" on page 248 for information on how to configure huge pages on a Wind River node and see "Configure the Number of Huge Pages to Use" on page 403 for information on how to configure the number of huge pages that the Cloud-Native Router vRouter uses.
dpdk_monitoring_thread_config	(Optional) Enables a monitoring thread for the vRouter DPDK container. Every `loggingInterval` seconds, a log containing the information indicated by `loggingMask` is generated.
loggingMask	Specifies the information to be generated. Represented by a bitmask with bit positions as follows: • 0b001 is the nl_counter • 0b010 is the lcore_timestamp • 0b100 is the profile_histogram
loggingInterval	Specifies the log generation interval in seconds.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
ddp	(Optional) Indicates the global Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at the 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. See *Enabling Dynamic Device Personalization (DDP) on Individual Interfaces* for more details. Options include auto, on, or off. Default is off. **NOTE**: The interface level `ddp` takes precedence over the global `ddp` configuration.
twampPort	(Optional) The TWAMP session reflector port (if you want TWAMP sessions to use vRouter timestamps). The vRouter listens to TWAMP test messages on this port and inserts/ overwrites timestamps in TWAMP test messages. Timestamping TWAMP messages at the vRouter (instead of at cRPD) leads to more accurate measurements. Valid values are 862 and 49152 through 65535. If this parameter is absent, then the vRouter does not insert or overwrite timestamps in the TWAMP session. Timestamps are taken and inserted by cRPD instead. See *Two-Way Active Measurement Protocol (TWAMP)*.
vrouter_dpdk_uio_driver	The uio driver is `vfio-pci`.
agentModeType	Set to dpdk.
fabricRpfCheckDisable	Set to false to enable the RPF check on all Cloud-Native Router fabric interfaces. By default, RPF check is disabled.
telemetry	(Optional) Configures cRPD telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
disable	Set to true to disable cRPD telemetry. Default is false, which means that cRPD telemetry is enabled by default.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
metricsPort	The port that the cRPD telemetry exporter is listening to Prometheus queries on. Default is 8072.
logLevel	One of warn, warning, info, debug, trace, or verbose. Default is info.
gnmi	(Optional) Configures cRPD gNMI settings.
	enable Set to true to enable the cRPD telemetry exporter to respond to gNMI requests.
vrouter	
telemetry	(Optional) Configures vRouter telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
	metricsPort Specify the port that the vRouter telemetry exporter listens to Prometheus queries on. Default is 8070.
	logLevel One of warn, warning, info, debug, trace, or verbose. Default is info.
	gnmi (Optional) Configures vRouter gNMI settings. enable - Set to true to enable the vRouter telemetry exporter to respond to gNMI requests.
persistConfig	Set to true if you want Cloud-Native Router pod configuration to persist even after uninstallation. This option can only be set for L2 mode deployments. Default is false.

Table 31: Helm Chart Description for Wind River Deployment *(Continued)*

Key	Description
enableLocalPersistence	Set to true if you're using the cRPD CLI or NETCONF to configure JCNR. When set to true, the cRPD CLI and NETCONF configuration persists through node reboots, cRPD pod restarts, and Cloud-Native Router upgrades. Default is false.
interfaceBoundType	Set to 1 to indicate a pre-bound SR-IOV interface. Default is 0.
networkDetails	Configures attributes related to the network attachment definitions.
ddp	Options are on or off. Default is off.
name	Specify the name of the network attachment definition.
namespace	Specify the namespace where the network attachment definition is created.
networkResources	Configures network device resources for the network attachment definitions.
limits	Set the limit for the number of SR-IOV interfaces used for each network attachment definition.
requests	Set the requested number of SR-IOV interfaces for each network attachment definition.
contrail-tools	
install	Set to true to install contrail-tools (used for debugging).

Customize Cloud-Native Router Configuration

SUMMARY

Read this topic to understand how to customize Cloud-Native Router configuration using a Configlet custom resource.

IN THIS SECTION

Configlet Custom Resource

Starting with Juniper Cloud-Native Router (JCNR) Release 24.2, we support customizing Cloud-Native Router configuration using a configlet custom resource. The configlet can be generated either by rendering a predefined template of supported Junos configuration or using raw configuration. The generated configuration is validated and deployed on the Cloud-Native Router controller (cRPD) as one or more Junos configuration groups.

NOTE: You can configure Cloud-Native Router using either configlets or the cRPD CLI or NETCONF. If you use the cRPD CLI or NETCONF, be sure to enable local persistence in **values.yaml** (`enableLocalPersistence: true`) so that your CLI or NETCONF configuration persists across reboots and upgrades.

NOTE: Using both configlets and the cRPD CLI or NETCONF to configure Cloud-Native Router may lead to unpredictable behavior. Use one or the other, but not both.

Configuration Examples

You create a configlet custom resource of the kind `Configlet` in the `jcnr` namespace. You provide raw configuration as Junos `set` commands.

Use `crpdSelector` to control where the configlet applies. The generated configuration is deployed to cRPD pods on nodes matching the specified label only. If `crpdSelector` is not defined, the configuration is applied to all cRPD pods in the cluster.

An example configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample             # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker                   # <-- Node label to select the cRPD pods
```

You can also use a templatized configlet yaml that contains keys or variables. The values for variables are provided by a configletDataValue custom resource, referenced by `configletDataValueRef` . An example templatized configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-with-template      # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: worker                          # <-- Node label to select the cRPD pods
  configletDataValueRef:
    name: "configletdatavalue-sample"       # <-- Configlet Data Value resource name
```

To render configuration using the template, you must provide key:value pairs in the ConfigletDataValue custom resource:

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
```

```
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"           # <-- Key:Value pair
  }
```

The generated configuration is validated and applied to all or selected cRPD pods as a Junos Configuration Group.

Applying the Configlet Resource

The configlet resource can be used to apply configuration to selected or all cRPD pods either when Cloud-Native Router is deployed or once the cRPD pods are up and running. Let us look at configlet deployment in detail.

Applying raw configuration

1. Create raw configuration configlet yaml. The example below configures a loopback interface in cRPD.

```
cat configlet-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker
```

2. Apply the configuration using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                                  AGE
configlet-sample-node1                10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

```
kubectl describe nodeconfiglet configlet-sample-node1 -n jcnr
```

   The following output has been trimmed for brevity:

```
Name:         configlet-sample-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
Annotations:  <none>
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample
```

```
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 10.10.10.1/32;
            }
        }
    }
}
```

NOTE: The configuration generated using configlets is applied to cRPD as configuration groups. We therefore recommend that you not use configuration groups when specifying your configlet.

Applying templatized configuration

1. Create the templatized configlet yaml and the configlet data value yaml for key:value pairs.

```
cat configlet-sample-template.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-template
  namespace: jcnr
```

```
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: master
  configletDataValueRef:
    name: "configletdatavalue-sample"
```

```
cat configletdatavalue-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"
  }
```

2. Apply the configuration using the `kubectl apply` command, starting with the config data value yaml.

```
kubectl apply -f configletdatavalue-sample.yaml
```

```
configletdatavalue.configplane.juniper.net/configletdatavalue-sample created
```

```
kubectl apply -f configlet-sample-template.yaml
```

```
configlet.configplane.juniper.net/configlet-sample-template created
```

3. Check on the configlet.

When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                                      AGE
configlet-sample-template-node1           10m
```

If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

For example:

```
kubectl describe nodeconfiglet configlet-sample-template-node1 -n jcnr
```

The following output has been trimmed for brevity:

```
Name:         configlet-sample-template-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
Annotations:  <none>
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample-template
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

   ```
   show configuration groups configlet-sample-template
   ```

   ```
   interfaces {
       lo0 {
           unit 0 {
               family inet {
                   address 127.0.0.1/32;
               }
           }
       }
   }
   ```

Modifying the Configlet

You can modify a configlet resource by changing the yaml file and reapplying it using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample configured
```

Any changes to existing configlet resource are reconciled by replacing the configuration group on cRPD.

You can delete the configuration group by deleting the configlet resource using the `kubectl delete` command.

```
kubectl delete configlet configlet-sample -n jcnr
```

```
configlet.configplane.juniper.net "configlet-sample" deleted
```

Troubleshooting

If you run into problems, check the `contrail-k8s-deployer` logs. For example:

```
kubectl logs contrail-k8s-deployer-8ff895cc5-cbfwm -n contrail-deploy
```

7

CHAPTER

Install Cloud-Native Router on Microsoft Azure Cloud Platform

IN THIS CHAPTER

Install and Verify Juniper Cloud-Native Router for Azure Deployment

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

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

1. Review the "System Requirements for Azure Deployment" on page 293 section to ensure the setup has all the required configuration.
2. Download the desired Cloud-Native Router software package to the directory of your choice.

   You have the option of downloading the package to install Cloud-Native Router only or downloading the package to install JNCR together with Juniper cSRX. See "Cloud-Native Router Software Download Packages" on page 387 for a description of the packages available. If you don't want to install Juniper cSRX now, you can always choose to install Juniper cSRX on your working Cloud-Native Router installation later.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

   ```
   tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
   ```
4. Change directory to the main installation directory.
   - If you're installing Cloud-Native Router only, then:

     ```
     cd Juniper_Cloud_Native_Router_<release>
     ```

This directory contains the Helm chart for Cloud-Native Router only.

- If you're installing Cloud-Native Router and cSRX at the same time, then:

```
cd Juniper_Cloud_Native_Router_CSRX_<release>
```

This directory contains the combination Helm chart for Cloud-Native Router and cSRX.

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

5. View the contents in the current directory.

```
ls
helmchart  images  README.md  secrets
```

6. Change to the **helmchart** directory and expand the Helm chart.

```
cd helmchart
```

- For Cloud-Native Router only:

```
ls
jcnr-<release>.tgz
```

```
tar -xzvf jcnr-<release>.tgz
```

```
ls
jcnr   jcnr-<release>.tgz
```

The Helm chart is located in the **jcnr** directory.

- For the combined Cloud-Native Router and cSRX:

```
ls
jcnr_csrx-<release>.tgz
```

```
tar -xzvf jcnr_csrx-<release>.tgz
```

```
ls
jcnr_csrx   jcnr_csrx-<release>.tgz
```

The Helm chart is located in the **jcnr_csrx** directory.

7. 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 398 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 399 for instructions on how to import images to the local container runtime.

8. Follow the steps in "Installing Your License" on page 353 to install your Cloud-Native Router license.

9. 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**.

10. Apply **secrets/jcnr-secrets.yaml** to the cluster.

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

11. If desired, configure how cores are assigned to the vRouter DPDK containers. See "Allocate CPUs to the Cloud-Native Router Forwarding Plane" on page 355.
12. Customize the Helm chart for your deployment using the **helmchart/jcnr/values.yaml** or **helmchart/jcnr_csrx/values.yaml** file.

    See "Customize Cloud-Native Router Helm Chart for Azure Deployment" on page 302 for descriptions of the Helm chart configurations.
13. Optionally, customize Cloud-Native Router configuration.

    See, "Customize Cloud-Native Router Configuration " on page 62 for creating and applying the cRPD customizations.
14. If you're installing Juniper cSRX now, then follow the procedure in "Apply the cSRX License and Configure cSRX" on page 338.
15. Label the nodes where you want Cloud-Native Router to be installed based on the `nodeaffinity` configuration (if defined in the `values.yaml`). For example:

```
kubectl label nodes ip-10.0.100.17.lab.net key1=jcnr --overwrite
```

16. Deploy the Juniper Cloud-Native Router using the Helm chart.

    Navigate to the **helmchart/jcnr** or the **helmchart/jcnr_csrx** directory and run the following command:

```
helm install jcnr .
```

or

```
helm install jcnr-csrx .
```

```
NAME: jcnr
LAST DEPLOYED: Fri Dec 22 06:04:33 2023
NAMESPACE: default
STATUS: deployed
```

```
REVISION: 1
TEST SUITE: None
```

17. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

Sample output:

```
NAME        NAMESPACE    REVISION        UPDATED
STATUS          CHART                      APP VERSION
jcnr        default      1               2023-12-22 06:04:33.144611017 -0400 EDT
deployed        jcnr-<version>                 <version>
```

Verify Installation

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

NOTE: The output shown in this example procedure is affected by the number of nodes in the cluster. The output you see in your setup may differ in that regard.

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          jcnr-0-dp-contrail-vrouter-nodes-b2jxp        2/2     Running   0                87s
contrail          jcnr-0-dp-contrail-vrouter-nodes-vrdpdk-g7wrk 1/1     Running   0                87s
jcnr              jcnr-0-crpd-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   0               4h2m
kube-system      calico-node-28w98                              1/1     Running   0               86d
kube-system      coredns-54bf8d85c7-vkpgs                       1/1     Running   0               3h8m
kube-system      dns-autoscaler-7944dc7978-ws9fn                1/1     Running   0               86d
kube-system      kube-apiserver-ix-esx-06                       1/1     Running   0               86d
kube-system      kube-controller-manager-ix-esx-06              1/1     Running   0               86d
kube-system      kube-multus-ds-amd64-jl69w                     1/1     Running   0               86d
kube-system      kube-proxy-qm5bl                               1/1     Running   0               86d
kube-system      kube-scheduler-ix-esx-06                       1/1     Running   0               86d
kube-system      nodelocaldns-bntfp                             1/1     Running   0               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      jcnr-0-dp-contrail-vrouter-nodes           1         1         1       1            1
<none>                   90m
contrail      jcnr-0-dp-contrail-vrouter-nodes-vrdpdk    1         1         1       1            1
<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        jcnr-0-crpd   1/1     27m
```

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

   a. View the *Access cRPD CLI* 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 *Access vRouter CLI* 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: 1500
            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: 1500
```

```
            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: 1500
            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.

System Requirements for Azure Deployment

IN THIS SECTION

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

Minimum Host System Requirements for Azure

Table 32 on page 294 lists the host system requirements for installing Cloud-Native Router on Azure.

Table 32: Minimum Host System Requirements for Azure

Component	Value/Version	Notes
Azure Deployment	VM-based	
Instance Type	Standard_F16s_v2	
CPU	Intel x86	The tested CPU is Intel Cascade Lake
Host OS	Rocky Linux 8.7	
Kernel Version	Rocky Linux: 4.18.X	
Kubernetes (K8s)	Version 1.25.x	
Calico	Version 3.25.1	
Multus	Version 4.0	
Helm	3.9.x	
Container-RT	containerd 1.6.x, 1.7.x	Other container runtimes may work but have not been tested with JCNR.
NOTE: The component versions listed in this table are expected to work with JCNR, but not every version or combination is tested in every release.		

Resource Requirements for Azure

Table 33 on page 295 lists the resource requirements for installing Cloud-Native Router on Azure.

Table 33: Resource Requirements for Azure

Resource	Value	Usage Notes
Data plane forwarding cores	1 core (1P + 1S)	
Service/Control Cores	0	

UIO Driver	uio_hv_generic	To enable, add the following modules to be loaded at boot:
		`cat /etc/modules-load.d/k8s.conf`
		`uio`
		`uio_hv_genericib_uverbs`
		`mlx4_ib`
		The above libraries are provided by ibverbs package.

Hugepages (1G)	6 Gi	See "Configure the Number of Huge Pages Available on a Node" on page 401.
Cloud-Native Router Controller cores	.5	
Cloud-Native Router vRouter Agent cores	.5	

Miscellaneous Requirements for Azure

Table 34 on page 296 lists additional requirements for installing Cloud-Native Router on Azure.

Table 34: Miscellaneous Requirements for Azure

Requirement	Example

Set IOMMU and IOMMU-PT in GRUB.	Add the following line to **/etc/default/grub**.
	`GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
	Update grub and reboot.
	`grub2-mkconfig -o /boot/grub2/grub.cfg`
	`reboot`

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.
NOTE: Applicable for L3 deployments only.
	Create a **/etc/modules-load.d/crpd.conf** file and add the following kernel modules to it:
	`tun`
	`fou`
	`fou6`
	`ipip`
	`ip_tunnel`
	`ip6_tunnel`
	`mpls_gso`
	`mpls_router`
	`mpls_iptunnel`
	`vrf`
	`vxlan`

Enable kernel-based forwarding on the Linux host.	`ip fou add port 6635 ipproto 137`

Table 34: Miscellaneous Requirements for Azure *(Continued)*

Requirement	Example
Add firewall rules for loopback address for VPC.	Configure the VPC firewall rule to allow ingress traffic with source filters set to the subnet range to which Cloud-Native Router is attached, along with the IP ranges or addresses for the loopback addresses. For example: Navigate to Firewall policies on the Azure console and create a firewall rule with the following attributes: 1. `Name`: Name of the firewall rule 2. `Network`: Choose the VPC network 3. `Priority`: 1000 4. `Direction`: Ingress 5. `Action on Match`: Allow 6. `Source filters`: 10.2.0.0/24, 10.51.2.0/24, 10.51.1.0/24, 10.12.2.2/32, 10.13.3.3/32 7. `Protocols`: all 8. `Enforcement`: Enabled where 10.2.0.0/24 is the subnet to which Cloud-Native Router is attached and 10.51.2.0/24, 10.51.1.0/24, 10.12.2.2/32, and 10.13.3.3/32 are loopback IP ranges.
Set the MTU on all fabric interfaces to 1500 bytes.	After Cloud-Native Router comes up, use the cRPD CLI to set the MTU size on all fabric interfaces to 1500 bytes. Microsoft Azure Cloud Platform recommends an MTU size less than or equal to 1500 bytes on all interfaces that connect directly to the Azure infrastructure. These interfaces are the Cloud-Native Router fabric interfaces. Failure to follow this rule might lead to packet drops. For information on how to access the cRPD CLI, see *Access cRPD CLI*.

Table 34: Miscellaneous Requirements for Azure *(Continued)*

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td>Ensure accelerated networking is enabled for the fabric interface.</td><td>

If accelerated networking is enabled properly, two interfaces become available for the fabric interface. For example:

<pre><code>3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc mq state UP group default qlen 1000
    link/ether 00:22:48:23:3b:9e brd
ff:ff:ff:ff:ff:ff
    inet 10.225.0.6/24 brd 10.225.0.255 scope global
eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::222:48ff:fe23:3b9e/64 scope link
       valid_lft forever preferred_lft forever
4: enP22960s2:
<BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 1500
qdisc mq master eth1 state UP group default qlen 1000
    link/ether 00:22:48:23:3b:9e brd
ff:ff:ff:ff:ff:ff
    altname enP22960p0s2</code></pre>

When configuring the fabric interface in the Helm chart, you must provide the interface with `hv_netvsc` bound to it. Issue the `ethtool -i` *`interface_name`* command to verify it. For example:

<pre><code>user@jcnr01:~# ethtool -i eth1
driver: hv_netvsc
version: 5.15.0-1049-azure
firmware-version: N/A
expansion-rom-version:
bus-info:
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: yes
supports-priv-flags: no</code></pre>

NOTE: Do not enable accelerated networking for the management interface.

</td></tr>
</table>

Table 34: Miscellaneous Requirements for Azure *(Continued)*

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td>Exclude Cloud-Native Router interfaces from NetworkManager control.</td><td>
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 Kubernetes management and create problems.
To avoid NetworkManager from interfering with Cloud-Native Router interface configuration, exclude Cloud-Native Router interfaces from NetworkManager control. Here's an example on how to do this in some Linux distributions:

Create the /etc/NetworkManager/conf.d/crpd.conf file and list the interfaces that you don't want NetworkManager to manage.
For example:
<pre>[keyfile]
 unmanaged-devices+=interface-name:enp*;interface-
name:ens*</pre>
where <code>enp*</code> and <code>ens*</code> refer to your Cloud-Native Router interfaces.
NOTE: <code>enp*</code>
indicates all interfaces starting with
<code>enp</code>
. For specific interface names, provided a comma-separated list.
Restart the NetworkManager service:
<pre>sudo systemctl restart NetworkManager</pre>
Edit the /etc/sysctl.conf file on the host and paste the following content in it:
<pre>net.ipv6.conf.default.addr_gen_mode=0
net.ipv6.conf.all.addr_gen_mode=0</pre>

</td></tr>
</table>

Table 34: Miscellaneous Requirements for Azure *(Continued)*

Requirement	Example

	`net.ipv6.conf.default.autoconf=0`
	`net.ipv6.conf.all.autoconf=0`
	
	4. Run the command `sysctl -p /etc/sysctl.conf` to load the new **sysctl.conf** values on the host.

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`

NOTE: Cloud-Native Router supports only IPv4 for Azure.	

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 35: Cloud-Native Router Listening Ports

Protocol	Port	Description
TCP	8085	vRouter introspect–Used to gain internal statistical information about vRouter

Table 35: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	8070	Telemetry Information- Used to see telemetry data from the Cloud-Native Router vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from Cloud-Native Router control plane
TCP	8075, 8076	Telemetry Information- Used for gNMI requests
TCP	9091	vRouter health check–cloud-native router checks to ensure the vRouter agent is running.
TCP	9092	vRouter health check–cloud-native router checks to ensure the vRouter DPDK is running.
TCP	50052	gRPC port–Cloud-Native Router listens on both IPv4 and IPv6
TCP	8081	Cloud-Native Router Deployer Port
TCP	24	cRPD SSH
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	50053	Default gNMI port that listens to the client subscription request

Table 35: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

See "Cloud-Native Router Software Download Packages" on page 387.

NOTE: Before deploying Cloud-Native Router on Azure via Helm charts downloaded from the Juniper Networks software download site, you must whitelist the https://enterprise.hub.juniper.net URL as the Cloud-Native Router image repository.

Cloud-Native Router Licensing

See "Manage Cloud-Native Router Licenses" on page 352.

Customize Cloud-Native Router Helm Chart for Azure Deployment

IN THIS SECTION

Read this topic to learn about the deployment configuration available for the Juniper Cloud-Native Router when deployed on Microsoft Azure Cloud Platform.

You can deploy and operate Juniper Cloud-Native Router in L3 mode on Azure. You configure the deployment mode by editing the appropriate attributes in the `values.yaml` file prior to deployment.

Helm Chart Description for Azure Deployment

Customize the Helm chart using the **Juniper_Cloud_Native_Router_*<release>*/helmchart/jcnr/values.yaml** file. We provide a copy of the default **values.yaml** in "Cloud-Native Router Default Helm Chart" on page 389.

Table 36 on page 303 contains a description of the configurable attributes in **values.yaml** for an Azure deployment.

Table 36: Helm Chart Description for Azure Deployment

Key	Description
global	
installSyslog	Set to true to install syslog-ng.
registry	Defines the Docker registry for the Cloud-Native Router container images. The default value is `enterprise-hub.juniper.net`. The images provided in the tarball are tagged with the default registry name. If you choose to host the container images to a private registry, replace the default value with your registry URL.
repository	(Optional) Defines the repository path for the Cloud-Native Router container images. This is a global key that takes precedence over the `repository` paths under the `common` section. Default is `jcnr-container-prod/`.
imagePullSecret	(Optional) Defines the Docker registry authentication credentials. You can configure credentials to either the Juniper Networks enterprise-hub.juniper.net registry or your private registry.
registryCredentials	Base64 representation of your Docker registry credentials. See "Configure Repository Credentials" on page 398 for more information.
secretName	Name of the secret object that will be created.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description
common	Defines repository paths and tags for the Cloud-Native Router container images. Use defaults unless using a private registry.
repository	Defines the repository path. The default value is `jcnr-container-prod/`. 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.
readinessCheck	Set to true to enable Cloud-Native Router Readiness preflight and postflight checks during installation. Comment this out or set to false to disable Cloud-Native Router Readiness preflight and postflight checks. Preflight checks verify that your infrastructure can support JCNR. Preflight checks take place before Cloud-Native Router is installed. Postflight checks verify that your Cloud-Native Router installation is working properly. Postflight checks take place after Cloud-Native Router is installed. See "Cloud-Native Router Readiness Checks" on page 362.
replicas	(Optional) Indicates the number of replicas for cRPD. Default is 1. The value for this key must be specified for multi-node clusters. The value is equal to the number of nodes running JCNR.
noLocalSwitching	Not applicable.
iamRole	Not applicable.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description

fabricInterface	Provide a list of interfaces to be bound to the DPDK.
	
	NOTE: Use the L3 only section to configure fabric interfaces for Azure. The L2 only and L2-L3 sections are not applicable for Azure deployments. Do not configure `interface_mode` for any of the interfaces.
	
	For example:
	
	`# L3  only`
	`- eth1:`
	`    ddp: "off"`
	`- eth2:`
	`    ddp: "off"`
	
	See "Cloud-Native Router Interfaces Overview" on page 14 for more information.

subnet	Not applicable.
ddp	Not applicable.
interface_mode	Not applicable.
vlan-id-list	Not applicable.
storm-control-profile	Not applicable.
native-vlan-id	Not applicable.
no-local-switching	Not applicable.
qosSchedulerProfileName	Specifies the QoS scheduler profile applicable to this interface. See the `qosSchedulerProfiles` section. If you don't specify a profile, then the QoS scheduler is disabled for this interface, which means that packets are scheduled with no regard to traffic class.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description
fabricWorkloadInterface	Not applicable.
log_level	Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave it set to the default INFO unless instructed to change it by Juniper Networks support.
log_path	The defined directory stores various JCNR-related descriptive logs such as **contrail-vrouter-agent.log**, **contrail-vrouter-dpdk.log**, etc. Default is **/var/log/jcnr/**.
syslog_notifications	Indicates the absolute path to the file that stores `syslog-ng` generated notifications in JSON format. Default is **/var/log/jcnr/jcnr_notifications.json**.
corePattern	Indicates the `core_pattern` for the core file. If left blank, then Cloud-Native Router pods will not overwrite the default pattern on the host. **NOTE**: Set the `core_pattern` on the host before deploying JCNR. You can 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 to the `core` file. Default is **/var/crash**.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description

nodeAffinity	(Optional) Defines labels on nodes to determine where to place the vRouter pods.
	
	By default the vRouter pods are deployed to all nodes of a cluster.
	
	In the example below, the node affinity label is defined as `key1=jcnr`. You must apply this label to each node where Cloud-Native Router is to be deployed:
	
	`nodeAffinity:`
	`- key: key1`
	`operator: In`
	`values:`
	`- jcnr`
	
	NOTE: This key is a global setting.

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 path with the path in your distribution (for example, **/var/opt/cni/bin**).
grpcTelemetryPort	(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50053.
grpcVrouterPort	(Optional) Default is 50052. Configure to override.
vRouterDeployerPort	(Optional) Default is 8081. Configure to override.
jcnr-vrouter	

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description
cpu_core_mask	If present, this indicates that you want to use static CPU allocation to allocate cores to the forwarding plane. This value should be a comma-delimited list of isolated CPU cores that you want to statically allocate to the forwarding plane (for example, `cpu_core_mask: "2,3,22,23"`). Use the cores not used by the host OS. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Cloud-Native Router Readiness preflight checks, if enabled, will fail the installation if you specify both.
guaranteedVrouterCpus	If present, this indicates that you want to use the Kubernetes CPU Manager to allocate CPU cores to the forwarding plane. This value should be the number of guaranteed CPU cores that you want the Kubernetes CPU Manager to allocate to the forwarding plane. You should set this value to at least one more than the number of forwarding cores. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane. **NOTE**: You cannot use static CPU allocation and Kubernetes CPU Manager at the same time. Using both can lead to unpredictable behavior.
dpdkCtrlThreadMask	Specifies the CPU core(s) to allocate to vRouter DPDK control threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `serviceCoreMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description
serviceCoreMask	Specifies the CPU core(s) to allocate to vRouter DPDK service threads when using static CPU allocation. This list should be a subset of the cores listed in `cpu_core_mask` and can be the same as the list in `dpdkCtrlThreadMask`. CPU cores listed in `cpu_core_mask` but not in `serviceCoreMask` or `dpdkCtrlThreadMask` are allocated for forwarding. Comment this out if you want to use Kubernetes CPU Manager to allocate cores to the forwarding plane.
numServiceCtrlThreadCPU	Specifies the number of CPU cores to allocate to vRouter DPDK service/control traffic when using the Kubernetes CPU Manager. This number should be smaller than the number of `guaranteedVrouterCpus` cores. The remaining `guaranteedVrouterCpus` cores are allocated for forwarding. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
numberOfSchedulerLcores	The number of CPU cores that you want Kubernetes CPU Manager to dedicate to your QoS schedulers. Comment this out if you want to use static CPU allocation to allocate cores to the forwarding plane.
restoreInterfaces	Set to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts or if Cloud-Native Router is uninstalled.
bondInterfaceConfigs	Not applicable.
mtu	Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default is 9000.
qosSchedulerProfiles	Defines the QoS scheduler profiles that are referenced from the `fabricInterface` section.
sched_profile_1	The name of the QoS scheduler profile.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description
	cpu Specify the CPU core(s) to dedicate to the scheduler. If `cpu_core_mask` is specified, this should be a unique additional core(s). **bandwidth** Specify the bandwidth in Gbps.
stormControlProfiles	Not applicable.
dpdkCommandAdditionalArgs	Pass any additional DPDK command line parameters. The `--yield_option 0` is set by default and implies the DPDK forwarding cores will not yield their assigned CPU cores. Other common parameters that can be added are `tx` and `rx` descriptors and `mempool`. For example: `dpdkCommandAdditionalArgs: "--yield_option 0 --dpdk_txd_sz 2048 --dpdk_rxd_sz 2048 --vr_mempool_sz 131072"` **NOTE**: Changing the number of `tx` and `rx` descriptors and the `mempool` size affects the number of huge pages required. If you make explicit changes to these parameters, set the number of huge pages to 10 (x 1 GB). See "Configure Huge Pages" on page 401 for information on how to configure huge pages.
dpdk_monitoring_thread_config	(Optional) Enables a monitoring thread for the vRouter DPDK container. Every `loggingInterval` seconds, a log containing the information indicated by `loggingMask` is generated.
loggingMask	Specifies the information to be generated. Represented by a bitmask with bit positions as follows: • 0b001 is the nl_counter • 0b010 is the lcore_timestamp • 0b100 is the profile_histogram
loggingInterval	Specifies the log generation interval in seconds.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description
ddp	Not applicable.
twampPort	(Optional) The TWAMP session reflector port (if you want TWAMP sessions to use vRouter timestamps). The vRouter listens to TWAMP test messages on this port and inserts/ overwrites timestamps in TWAMP test messages. Timestamping TWAMP messages at the vRouter (instead of at cRPD) leads to more accurate measurements. Valid values are 862 and 49152 through 65535. If this parameter is absent, then the vRouter does not insert or overwrite timestamps in the TWAMP session. Timestamps are taken and inserted by cRPD instead. See *Two-Way Active Measurement Protocol (TWAMP)*.
vrouter_dpdk_uio_driver	The uio driver is `uio_hv_generic`.
agentModeType	Set to dpdk.
fabricRpfCheckDisable	Set to false to enable the RPF check on all Cloud-Native Router fabric interfaces. By default, RPF check is disabled.
telemetry	(Optional) Configures cRPD telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
disable	Set to true to disable cRPD telemetry. Default is false, which means that cRPD telemetry is enabled by default.
metricsPort	The port that the cRPD telemetry exporter is listening to Prometheus queries on. Default is 8072.
logLevel	One of warn, warning, info, debug, trace, or verbose. Default is info.
gnmi	(Optional) Configures cRPD gNMI settings.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description
	enable Set to true to enable the cRPD telemetry exporter to respond to gNMI requests.
vrouter	
telemetry	(Optional) Configures vRouter telemetry settings. To learn more about telemetry, see *Telemetry Capabilities* .
	metricsPort Specify the port that the vRouter telemetry exporter listens to Prometheus queries on. Default is 8070.
	logLevel One of warn, warning, info, debug, trace, or verbose. Default is info.
	gnmi (Optional) Configures vRouter gNMI settings. enable - Set to true to enable the vRouter telemetry exporter to respond to gNMI requests.
persistConfig	Set to true if you want Cloud-Native Router pod configuration to persist even after uninstallation. This option can only be set for L2 mode deployments. Default is false.
enableLocalPersistence	Set to true if you're using the cRPD CLI or NETCONF to configure JCNR. When set to true, the cRPD CLI and NETCONF configuration persists through node reboots, cRPD pod restarts, and Cloud-Native Router upgrades. Default is false.
interfaceBoundType	Not applicable.
networkDetails	Not applicable.
networkResources	Not applicable.

Table 36: Helm Chart Description for Azure Deployment *(Continued)*

Key	Description
contrail-tools	
install	Set to true to install contrail-tools (used for debugging).

Customize Cloud-Native Router Configuration

SUMMARY

Read this topic to understand how to customize Cloud-Native Router configuration using a Configlet custom resource.

IN THIS SECTION

Configlet Custom Resource

Starting with Juniper Cloud-Native Router (JCNR) Release 24.2, we support customizing Cloud-Native Router configuration using a configlet custom resource. The configlet can be generated either by rendering a predefined template of supported Junos configuration or using raw configuration. The generated configuration is validated and deployed on the Cloud-Native Router controller (cRPD) as one or more Junos configuration groups.

NOTE: You can configure Cloud-Native Router using either configlets or the cRPD CLI or NETCONF. If you use the cRPD CLI or NETCONF, be sure to enable local persistence in **values.yaml** (`enableLocalPersistence: true`) so that your CLI or NETCONF configuration persists across reboots and upgrades.

NOTE: Using both configlets and the cRPD CLI or NETCONF to configure Cloud-Native Router may lead to unpredictable behavior. Use one or the other, but not both.

Configuration Examples

You create a configlet custom resource of the kind `Configlet` in the `jcnr` namespace. You provide raw configuration as Junos `set` commands.

Use `crpdSelector` to control where the configlet applies. The generated configuration is deployed to cRPD pods on nodes matching the specified label only. If `crpdSelector` is not defined, the configuration is applied to all cRPD pods in the cluster.

An example configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample            # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker                    # <-- Node label to select the cRPD pods
```

You can also use a templatized configlet yaml that contains keys or variables. The values for variables are provided by a configletDataValue custom resource, referenced by `configletDataValueRef` . An example templatized configlet yaml is provided below:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-with-template      # <-- Configlet resource name
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
```

```
  crpdSelector:
    matchLabels:
      node: worker                           # <-- Node label to select the cRPD pods
  configletDataValueRef:
    name: "configletdatavalue-sample"        # <-- Configlet Data Value resource name
```

To render configuration using the template, you must provide key:value pairs in the ConfigletDataValue custom resource:

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"          # <-- Key:Value pair
  }
```

The generated configuration is validated and applied to all or selected cRPD pods as a Junos Configuration Group.

Applying the Configlet Resource

The configlet resource can be used to apply configuration to selected or all cRPD pods either when Cloud-Native Router is deployed or once the cRPD pods are up and running. Let us look at configlet deployment in detail.

Applying raw configuration

1. Create raw configuration configlet yaml. The example below configures a loopback interface in cRPD.

```
cat configlet-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
```

```
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address 10.10.10.1/32
  crpdSelector:
    matchLabels:
      node: worker
```

2. Apply the configuration using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                          AGE
configlet-sample-node1        10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

```
kubectl describe nodeconfiglet configlet-sample-node1 -n jcnr
```

   The following output has been trimmed for brevity:

```
Name:         configlet-sample-node1
Namespace:    jcnr
Labels:       core.juniper.net/nodeName=node1
Annotations:  <none>
```

```
API Version:  configplane.juniper.net/v1
Kind:         NodeConfiglet
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 10.10.10.1/32;
            }
        }
    }
}
```

NOTE: The configuration generated using configlets is applied to cRPD as configuration groups. We therefore recommend that you not use configuration groups when specifying your configlet.

Applying templatized configuration

1. Create the templatized configlet yaml and the configlet data value yaml for key:value pairs.

```
cat configlet-sample-template.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample-template
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0 unit 0 family inet address {{ .Ip }}
  crpdSelector:
    matchLabels:
      node: master
  configletDataValueRef:
    name: "configletdatavalue-sample"
```

```
cat configletdatavalue-sample.yaml
```

```
apiVersion: configplane.juniper.net/v1
kind: ConfigletDataValue
metadata:
  name: configletdatavalue-sample
  namespace: jcnr
spec:
  data: {
    "Ip": "127.0.0.1"
  }
```

2. Apply the configuration using the `kubectl apply` command, starting with the config data value yaml.

```
kubectl apply -f configletdatavalue-sample.yaml
```

```
configletdatavalue.configplane.juniper.net/configletdatavalue-sample created
```

```
kubectl apply -f configlet-sample-template.yaml
```

```
configlet.configplane.juniper.net/configlet-sample-template created
```

3. Check on the configlet.

   When a configlet resource is deployed, it creates additional node configlet custom resources, one for each node matched by the `crpdSelector`.

```
kubectl get nodeconfiglets -n jcnr
```

```
NAME                                       AGE
configlet-sample-template-node1            10m
```

   If the configuration defined in the configlet yaml is invalid or fails to deploy, you can view the error message using `kubectl describe` for the node configlet custom resource.

   For example:

```
kubectl describe nodeconfiglet configlet-sample-template-node1 -n jcnr
```

   The following output has been trimmed for brevity:

```
Name:          configlet-sample-template-node1
Namespace:     jcnr
Labels:        core.juniper.net/nodeName=node1
Annotations:   <none>
API Version:   configplane.juniper.net/v1
Kind:          NodeConfiglet
```

```
Metadata:
  Creation Timestamp:  2024-06-13T16:51:23Z
 ...
Spec:
  Clis:
    set interfaces lo0 unit 0 address 10.10.10.1/32
  Group Name:  configlet-sample-template
  Node Name:   node1
Status:
  Message:  load-configuration failed: syntax error
  Status:   False
Events:     <none>
```

4. Optionally, verify the configuration on the *Access cRPD CLI* shell in CLI mode. Note that the configuration is applied as a configuration group named after the configlet resource.

```
show configuration groups configlet-sample-template
```

```
interfaces {
    lo0 {
        unit 0 {
            family inet {
                address 127.0.0.1/32;
            }
        }
    }
}
```

Modifying the Configlet

You can modify a configlet resource by changing the yaml file and reapplying it using the `kubectl apply` command.

```
kubectl apply -f configlet-sample.yaml
```

```
configlet.configplane.juniper.net/configlet-sample configured
```

Any changes to existing configlet resource are reconciled by replacing the configuration group on cRPD.

You can delete the configuration group by deleting the configlet resource using the `kubectl delete` command.

```
kubectl delete configlet configlet-sample -n jcnr
```

```
configlet.configplane.juniper.net "configlet-sample" deleted
```

Troubleshooting

If you run into problems, check the `contrail-k8s-deployer` logs. For example:

```
kubectl logs contrail-k8s-deployer-8ff895cc5-cbfwm -n contrail-deploy
```

8

CHAPTER

Install Cloud-Native Router on VMWare Tanzu

IN THIS CHAPTER

Install and Verify Juniper Cloud-Native Router for VMWare Tanzu

The procedure for installing and verifying Cloud-Native Router on VMWare Tanzu is the same as the procedure for installing and verifying Cloud-Native Router on baremetal.

See "Install and Verify Juniper Cloud-Native Router for Baremetal Servers" on page 28.

System Requirements for Tanzu Deployment

IN THIS SECTION

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

Minimum Host System Requirements for Tanzu

Table 37 on page 324 lists the host system requirements for installing Cloud-Native Router on Tanzu.

Table 37: Minimum Host System Requirements for Tanzu

Component	Value/Version	Notes
CPU	Intel x86	The tested CPU is Intel Xeon Gold 6212U 24-core @2.4 GHz
Host OS (for TKG 2.3)	Photon OS 3.0	
Kernel Version (for TKG 2.3)	4.19.x	
NIC	• Intel E810 CVL with Firmware 4.22 0x8001a1cf 1.3346.0 • Intel E810 CPK with Firmware 2.20 0x80015dc1 1.3083.0 • Intel E810-CQDA2 with Firmware 4.20 0x80017785 1.3346.0 • Intel XL710 with Firmware 9.20 0x8000e0e9 0.0.0 • Mellanox ConnectX-6 • Mellanox ConnectX-7	Support for Mellanox NICs is considered a Juniper Technology Preview ("Tech Preview" on page 452) feature. When using Mellanox NICs, ensure your interface names do not exceed 11 characters in length.
IAVF driver	4.8.2	
ICE_COMMS	1.3.35.0	
ICE	1.11.20.13	ICE driver is used only with the Intel E810 NIC

Table 37: Minimum Host System Requirements for Tanzu *(Continued)*

Component	Value/Version	Notes
i40e	2.22.18.1	i40e driver is used only with the Intel XL710 NIC
Kubernetes (TKG 2.3)	1.23.8+vmware.3	
Calico	3.22.x	
Multus	3.8	
Helm	3.9.x	
Container-RT (TKG 2.3)	containerd 1.6.6x	
NOTE: The component versions listed in this table are expected to work with JCNR, but not every version or combination is tested in every release.		

Resource Requirements for Tanzu

Table 38 on page 325 lists the resource requirements for installing Cloud-Native Router on Tanzu.

Table 38: Resource Requirements for Tanzu

Resource	Value	Usage Notes
Data plane forwarding cores	1 core (1P + 1S)	
Service/Control Cores	0	

Table 38: Resource Requirements for Tanzu ***(Continued)***

Resource	Value	Usage Notes

UIO Driver	VFIO-PCI	To enable, follow the steps below:
		`cat /etc/modules-load.d/vfio.conf`
		`vfio`
		`vfio-pci`

Hugepages (1G)	6 Gi	See "Configure the Number of Huge Pages Available on a Node" on page 401.
Cloud-Native Router Controller cores	.5	
Cloud-Native Router vRouter Agent cores	.5	

Miscellaneous Requirements for Tanzu

Table 39 on page 326 lists additional requirements for installing Cloud-Native Router on Tanzu.

Table 39: Miscellaneous Requirements for Tanzu

Requirement	Example
Enable the host with SR-IOV and VT-d in the system's BIOS.	Depends on BIOS.

Table 39: Miscellaneous Requirements for Tanzu *(Continued)*

Requirement	Example

Enable VLAN driver at system boot.	Configure **/etc/modules-load.d/vlan.conf** as follows:
	`cat /etc/modules-load.d/vlan.conf`
	`8021q`
	Reboot and verify by executing the command:
	`lsmod | grep 8021q`

Enable VFIO-PCI driver at system boot.	Configure **/etc/modules-load.d/vfio.conf** as follows:
	`cat /etc/modules-load.d/vfio.conf`
	`vfio`
	`vfio-pci`
	Reboot and verify by executing the command:
	`lsmod | grep vfio`

Set IOMMU and IOMMU-PT in GRUB.	Add the following line to **/etc/default/grub**.
	`GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
	Update grub and reboot.
	`grub2-mkconfig -o /boot/grub2/grub.cfg`
	`reboot`

Table 39: Miscellaneous Requirements for Tanzu *(Continued)*

Requirement	Example

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.
 	
NOTE: Applicable for L3 deployments only.
	Create a **/etc/modules-load.d/crpd.conf** file and add the following kernel modules to it:
	
	`tun`
	`fou`
	`fou6`
	`ipip`
	`ip_tunnel`
	`ip6_tunnel`
	`mpls_gso`
	`mpls_router`
	`mpls_iptunnel`
	`vrf`
	`vxlan`

Enable kernel-based forwarding on the Linux host.	`ip fou add port 6635 ipproto 137`

Table 39: Miscellaneous Requirements for Tanzu *(Continued)*

Requirement	Example

Exclude Cloud-Native Router interfaces from NetworkManager control.	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 Kubernetes management and create problems.
	
	To avoid NetworkManager from interfering with Cloud-Native Router interface configuration, exclude Cloud-Native Router interfaces from NetworkManager control. Here's an example on how to do this in some Linux distributions:
	
	1. Create the **/etc/NetworkManager/conf.d/crpd.conf** file and list the interfaces that you don't want NetworkManager to manage.
	For example:
	`[keyfile]`
	` unmanaged-devices+=interface-name:enp*;interface-name:ens*`
	where `enp*` and `ens*` refer to your Cloud-Native Router interfaces.
	NOTE: `enp*` indicates all interfaces starting with `enp`. For specific interface names, provided a comma-separated list.
	2. Restart the NetworkManager service:
	`sudo systemctl restart NetworkManager`
	.
	3. Edit the **/etc/sysctl.conf** 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`

Table 39: Miscellaneous Requirements for Tanzu *(Continued)*

<table>
<tr><th>Requirement</th><th>Example</th></tr>
<tr><td></td><td><pre>net.ipv6.conf.default.autoconf=0
net.ipv6.conf.all.autoconf=0</pre>
4. Run the command <code>sysctl -p /etc/sysctl.conf</code> to load the new sysctl.conf values on the host.
5. Create the bond interface manually. For example:
<pre>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</pre></td></tr>
<tr><td>Verify the core_pattern value is set on the host before deploying JCNR.</td><td><pre>sysctl kernel.core_pattern
kernel.core_pattern = |/usr/lib/systemd/systemd-
coredump %P %u %g %s %t %c %h %e</pre>
You can update the core_pattern in <code>/etc/sysctl.conf</code>. For example:
<pre>kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_
%t.gz</pre></td></tr>
<tr><td>Enable iommu unsafe interrupts and unsafe noiommu mode.</td><td><pre>echo Y > /sys/module/vfio_iommu_type1/parameters/
allow_unsafe_interrupts</pre>
<pre>echo Y > /sys/module/vfio/parameters/
enable_unsafe_noiommu_mode</pre></td></tr>
</table>

Table 39: Miscellaneous Requirements for Tanzu *(Continued)*

Requirement	Example

Configure iptables to accept specified traffic.	`iptables -I INPUT -p tcp --dport 830 -j ACCEPT`
	`iptables -I INPUT -p tcp --dport 24 -j ACCEPT`
	`iptables -I INPUT -p tcp --dport 8085 -j ACCEPT`
	`iptables -I INPUT -p tcp --dport 8070 -j ACCEPT`
	
	`iptables -I INPUT -p tcp --dport 8072 -j ACCEPT`
	`iptables -I INPUT -p tcp --dport 50053 -j ACCEPT`
	
	`iptables -A INPUT -p icmp -j ACCEPT`
	`iptables -A OUTPUT -p icmp -j ACCEPT`
	
	`iptables -A INPUT   -s 224.0.0.0/4 -j ACCEPT`
	`iptables -A FORWARD -s 224.0.0.0/4 -d 224.0.0.0/4 -j ACCEPT`
	`iptables -A OUTPUT  -d 224.0.0.0/4 -j ACCEPT`

On the ESXi Hypervisor, enable 16 queues.	`set esxcli system module parameters set -m icen -p NumQPsPerVF=16,16,16,16`

On the ESXi Hypervisor, enable trust and disable spoofcheck:	`esxcli intnet sriovnic vf set -s false -t true  -v 0  -n vmnic2`
	
	Check the settings:
	
	`esxcli intnet sriovnic vf get -n vmnic2`
	
	`VF ID     Trusted     Spoof Check`
	`0         true        false`

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 40: 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 the Cloud-Native Router vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from Cloud-Native Router control plane
TCP	8075, 8076	Telemetry Information- Used for gNMI requests
TCP	9091	vRouter health check–cloud-native router checks to ensure the vRouter agent is running.
TCP	9092	vRouter health check–cloud-native router checks to ensure the vRouter DPDK is running.
TCP	50052	gRPC port–Cloud-Native Router listens on both IPv4 and IPv6
TCP	8081	Cloud-Native Router Deployer Port
TCP	24	cRPD SSH
TCP	830	cRPD NETCONF
TCP	666	**rpd**
TCP	1883	Mosquito mqtt–Publish/subscribe messaging utility
TCP	9500	**agentd** on cRPD

Table 40: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	21883	**na-mqttd**
TCP	50053	Default gNMI port that listens to the client subscription request
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

See "Cloud-Native Router Software Download Packages" on page 387.

Cloud-Native Router Licensing

See "Manage Cloud-Native Router Licenses" on page 352.

Customize Cloud-Native Router Helm Chart for Tanzu Deployment

The way that you configure the installation Helm chart for Cloud-Native Router on VMWare Tanzu is the same as the way that you configure the installation Helm chart for Cloud-Native Router on baremetal servers.

See "Customize Cloud-Native Router Helm Chart for Bare Metal Servers" on page 48.

Customize Cloud-Native Router Configuration

The procedure for customizing cRPD for Cloud-Native Router on VMWare Tanzu is the same as the procedure for customizing cRPD for Cloud-Native Router on baremetal.

See "Customize Cloud-Native Router Configuration " on page 62.

9

CHAPTER

Deploying Service Chain (cSRX) with JCNR

IN THIS CHAPTER

Deploying Service Chain (cSRX) with JCNR

SUMMARY

Read this section to learn how to customize and deploy a security services instance (cSRX) with the Cloud-Native Router.

IN THIS SECTION

You can integrate the Juniper Cloud-Native Router (JCNR) with Juniper's containerized SRX (cSRX) platform to provide security services such as IPsec. Using host-based service chaining, the cloud-native router is chained with a security service instance (cSRX) in the same Kubernetes cluster. The cSRX instance runs as a pod service in L3 mode. The cSRX instance is customized and deployed via a Helm chart.

You have the option of deploying Juniper cSRX when you're installing Cloud-Native Router or after you've installed JCNR. See "Cloud-Native Router Software Download Packages" on page 387 for a description of the available packages.

Install cSRX on an Existing Cloud-Native Router Installation

Follow this procedure to install a cSRX instance on an existing Cloud-Native Router installation. Ensure all Cloud-Native Router components are up and running before you start this procedure.

1. Download and expand the software package for installing Juniper cSRX on an existing Cloud-Native Router installation. See "Cloud-Native Router Software Download Packages" on page 387 for a description of the software packages available.

   ```
   tar -xzvf junos_csrx_<release>.tar.gz
   ```

2. Change to the **junos_csrx_*<release>*/helmchart** directory and expand the Helm chart.

```
cd junos_csrx_<release>/helmchart
```

```
ls
junos-csrx-<release>.tgz
```

```
tar -xzvf junos-csrx-<release>.tgz
```

```
ls
junos-csrx    junos-csrx-<release>.tgz
```

The Helm chart is located in the **junos-csrx** directory.

3. The cSRX 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 398 for example instructions on how to configure repository credentials in Helm charts.

    - Configure your cluster to deploy images from the image tarball included in the downloaded cSRX software package. See "Deploy Prepackaged Images" on page 399 for example instructions on how to import images to the local containerd runtime.

4. Follow the steps in "Apply the cSRX License and Configure cSRX" on page 338 to apply your cSRX license and configure the cSRX Helm chart.

5. Install cSRX.

    Navigate to the **junos_csrx_*<release>*/helmchart/junos-csrx** directory and issue the following command to install the cSRX instance.

```
helm install junos-csrx .
```

Install cSRX During Cloud-Native Router Installation

Follow the steps in the respective Cloud-Native Router installation sections to install JCNR. One of the steps will refer you to "Apply the cSRX License and Configure cSRX" on page 338.

Apply the cSRX License and Configure cSRX

Follow this procedure to apply your cSRX license and configure Juniper cSRX.

The following steps assume you're in the **Juniper_Cloud_Native_Router_CSRX_*<release>*** directory if installing cSRX and Cloud-Native Router together, or in the **junos_csrx_*<release>*** directory if installing cSRX on an existing Cloud-Native Router installation.

1. Copy the cluster kubeconfig to all nodes where you want to install the Cloud-Native Router and cSRX combination.

   This step applies to both installing cSRX during Cloud-Native Router installation and installing cSRX on an existing Cloud-Native Router installation. If you don't perform this step, the installation may fail.

   a. Copy the cluster kubeconfig to a location of your choice on the target node.

      For example, the following copies the cluster kubeconfig from its default location at **~/.kube/config** to **/root/kubeconfig** on the target node:

      ```
      scp ~/.kube/config <worker-node-ip>:/root/kubeconfig
      ```

      where *<worker-node-ip>* is the IP address of a node where you want to install both Cloud-Native Router and cSRX. Repeat for all nodes where you want to install both Cloud-Native Router and cSRX.

      NOTE: The destination file path must be the same on all target nodes.

   b. After copying the kubeconfig to all target nodes, set `kubeConfigPath` in **values.yaml** to the destination file location.

      For example:

      ```
      kubeconfigpath: /root/kubeconfig
      ```

      See "Customize cSRX Helm Chart" on page 340 for information on the parameters in **values.yaml**.

2. Apply your Juniper cSRX license.

   a. If the **secrets/csrx-secrets.yaml** doesn't exist in your software package, create it with the contents below:

      ```
      apiVersion: v1
      kind: Secret
      ```

```
metadata:
  name: service-chain-instance
  namespace: jcnr
data:
  csrx_license: |
    <add your license in base64 format>
  csrx_root_password: <add your root password in base64 format>
```

b. Encode your license in base64.

   Copy your Juniper cSRX license file onto your host server and issue the command:

```
base64 -w 0 licenseFile
```

   The output of this command is your base64-encoded license.

c. Replace `<add your license in base64 format>` with your base64-encoded license.

> **NOTE**: You must obtain your license file from your account team and install it in the **secrets/csrx-secrets.yaml** file as instructed above. The `csrx-init` container performs a license check and proceeds only if the required secret `service-chain-instance` is found.

d. Encode your root password in base64. The root password is required for NETCONF access for telemetry.

   Encode your password as follows:

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

   The output of this command is your base64-encoded root password.

e. Replace `<add your root password in base64 format>` with your base64-encoded root password.

f. Apply the **secrets/csrx-secrets.yaml** to the Kubernetes cluster.

```
kubectl apply -f secrets/csrx-secrets.yaml
secret/service-chain-instance created
```

3. Configure the cSRX Helm chart.

- If you're installing cSRX at the same time you're installing JCNR, then you're configuring the `junos-csrx` section of the combination Helm chart in **Juniper_Cloud_Native_Router_CSRX_*<release>*/helmchart/jcnr_csrx/values.yaml**.

- If you're installing cSRX on an existing Cloud-Native Router installation, then you're configuring the `csrx` section of the standalone Helm chart in **junos_csrx_*<release>*/helmchart/junos-csrx/values.yaml**.

Refer to the cSRX parameter descriptions in "Customize cSRX Helm Chart" on page 340.

Customize cSRX Helm Chart

The cSRX service chaining instance is deployed via a Helm chart, either a standalone Helm chart or a combined Helm chart with JCNR. The deployment consists of two essential components:

- csrx-init: This is an init container that prepares the configuration for the main cSRX application. It extracts the necessary information from the **values.yaml** file, processes it, and generates the configuration data for cSRX. This ensures that the main cSRX application starts with a valid, up-to-date configuration.

- csrx: The csrx is the main application container and the core component of the cSRX deployment. It relies on the configuration provided by the `csrx-init` container to function correctly.

You can customize the cSRX deployment by specifying a range of configuration parameters in the **values.yaml** file. Key configuration options include:

- kubeConfigPath: This is the path to the cluster kubeconfig file on the node(s) where you're installing Cloud-Native Router and cSRX. You copied the cluster kubeconfig to this file location on this node(s) in step 1 in "Apply the cSRX License and Configure cSRX" on page 338. If this parameter is commented out, then the cluster kubeconfig is assumed to be at **/etc/kubernetes/kubelet.conf**.

- interfaceType: This is the type of interface on the cSRX to connect to JCNR. Must be set to `vhost` only.

- interfaceConfigs: This is an array defining the interface IP address, gateway address and optionally routes. The interface IP must match the `localAddress` element in the `ipSecTunnelConfigs` array. The routes should contain prefixes to steer decrypted traffic to Cloud-Native Router and reachability route for IPSec gateway.

- ipSecTunnelConfigs: This is an array defining the IPsec configuration details such as ike-phase1, proposal, policy and gateway configuration. Traffic selector should contain traffic that is expected to be encrypted.

- jcnr_config: This is an array defining the routes to be configured in Cloud-Native Router to steer traffic from Cloud-Native Router to cSRX and to steer IPsec traffic from the remote IPsec gateway to the cSRX to apply the security service chain.
- telemetry: Enable or disable telemetry.

Here is the default **values.yaml** for standalone cSRX deployment:

```
# Default values for cSRX.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

common:
  registry: enterprise-hub.juniper.net/
  repository: jcnr-container-prod/

  csrxInit:
    repository:
    image: csrx-init
    tag: 24.4.0.196
    imagePullPolicy: IfNotPresent
    resources:
      #limits:
      #  memory: 1Gi
      #  cpu: 1
      #requests:
      #  memory: 1Gi
      #  cpu: 1



  csrx:
    repository:
    image: csrx
    tag: 24.4R1.9
    imagePullPolicy: IfNotPresent
    resources:
      limits:
        hugepages-1Gi: 4Gi
        memory: 4Gi
      requests:
        hugepages-1Gi: 4Gi
        memory: 4Gi
```

```
  csrxTelemetry:
    repository:
    image: contrail-telemetry-exporter
    tag: 24.4.0.196
    imagePullPolicy: IfNotPresent
    resources:

# 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

# nodeAffinity: Can be used to inject nodeAffinity for cSRX
# you may label the nodes where we wish to deploy cSRX and inject affinity accordingly
#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



replicas: 1

interfaceType: "vhost"

interfaceConfigs:
  #- name: eth1
  #  ip: 181.1.1.1/30          # should match ipSecTunnelConfigs localAddress if configured
  #  gateway: 181.1.1.2        # gateway configuration
  #  ip6: 181:1:1::1/64        # optional
  #  ip6Gateway: 181:1:1::2    # optional
  #  routes:                   # this field is optional
  #  - "191.1.1.0/24"
  #  - "200.1.1.0/24"
  #  instance_parameters:
```

```
  #    name: "untrust"
  #    type: "vrf"             # options include virtual-router or vrf
  #    vrfTarget: 10:10        # this option is valid only for vrf
  #- name: eth2
  #  ip: 1.21.1.1/30           # should match ipSecTunnelConfigs localAddress if configured
  #  gateway: 1.21.1.2         # gateway configuration
  #  ip6: 181:2:1::1/64                     # optional
  #  ip6Gateway: 181:2:1::2               # optional
  #  routes:                   # this field is optional
  #  - "111.1.1.0/24"
  #  - "192.1.1.0/24"
  #  instance_parameters:
  #    name: "trust"
  #    type: "vrf"             # options include virtual-router or vrf
  #    vrfTarget: 11:11        # this option is valid only for vrf

ipSecTunnelConfigs:       # untrust
  #- interface: eth1       ## section ike-phase1, proposal, policy, gateway
  #  gateway: 171.1.1.1
  #  localAddress: 181.1.1.1
  #  authenticationAlgorithm: sha-256
  #  encryptionAlgorithm: aes-256-cbc
  #  preSharedKey: "$9$zt3l3AuIRhev8FnNVsYoaApu0RcSyev8XO1NVYoDj.P5F9AyrKv8X"
  #  trafficSelector:
  #  - name: ts1
  #    localIP: 111.1.1.0/24 ## IP cannot be 0.0.0.0/0
  #    remoteIP: 222.1.1.0/24 ## IP cannot be 0.0.0.0/0

jcnr_config:
  #- name: eth1
  #  routes:
  #    - "121.1.1.0/24"

#csrx_flavor: specify the csrx deployment model. Corresponding values for csrx control and data
cpus
#must be provided based on the flavor mentioned below. Following are possible options:
# CSRX-2CPU-4G
# CSRX-4CPU-8G
# CSRX-6CPU-12G
# CSRX-8CPU-16G
# CSRX-16CPU-32G
# CSRX-20CPU-48G
csrx_flavor: CSRX-2CPU-4G
```

```
csrx_ctrl_cpu: "0x01"

csrx_data_cpu: "0x02"

telemetry:
  enable: false
  gnmi: true
  service:
    type: ClusterIP

    labels: {}
    annotations: {}
    clusterIP: ""

    # List of IP addresses at which the cSRX telemetry service is available
    # Ref: https://kubernetes.io/docs/user-guide/services/#external-ips
    externalIPs: []

    # Only use if service.type is "LoadBalancer"
    loadBalancerIP: ""

    # Ports to expose on each node
    # Only used if service.type is "NodePort"
    nodePort:
      prometheus: 30073
      gnmi: 30077
```

For a cSRX configuration example, see *IPsec Security Services* in the *Juniper Cloud Native Router User Guide*.

10

CHAPTER

Manage

IN THIS CHAPTER

Manage Cloud-Native Router Software

SUMMARY

This topic provides information on the available upgrade, downgrade and uninstall options for JCNR.

IN THIS SECTION

Upgrade from Cloud-Native Router Release 23.4 and Earlier

Upgrading to this release from Cloud-Native Router release 23.4 and earlier is not supported. You must uninstall your existing Cloud-Native Router release before you can install this Cloud-Native Router release. We show you how to do this below.

> **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 if your existing Cloud-Native Router release is earlier than release 23.2.

1. Save your current configuration.

   a. Save the Cloud-Native Router Helm chart **values.yaml** customizations that you made.

   b. Access the cRPD pods and save the Junos cRPD CLI configuration.

   To see the set of commands used to create the current configuration,

   ```
   show configuration | display set
   ```

   To save the configuration, use the Junos CLI `save` command.

2. Uninstall JCNR.

   See "Uninstall JCNR" on page 351 but don't delete the `jcnr` namespace or the `jcnr-secrets`.

3. Download the ***<sw_package>*.tar.gz** tarball to the directory of your choice. See "Cloud-Native Router Software Download Packages" on page 387 for the available package options.
4. Expand the downloaded package.

```
tar xzvf <sw_package>.tar.gz
```

5. Change directory to the main installation directory.

   If you're installing Cloud-Native Router only, then:

```
cd Juniper_Cloud_Native_Router_<release>
```

   This directory contains the Helm chart for Cloud-Native Router only.

   If you're installing Cloud-Native Router and cSRX at the same time, then:

```
cd Juniper_Cloud_Native_Router_CSRX_<release>
```

   This directory contains the combination Helm chart for both Cloud-Native Router and cSRX.

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

6. View the contents in the current directory.

```
ls
helmchart  images  README.md  secrets
```

7. Change to the **helmchart** directory and expand the Helm chart.

```
cd helmchart
```

```
ls
jcnr-<release>.tgz
```

```
tar -xzvf jcnr-<release>.tgz
```

```
ls
jcnr   jcnr-<release>.tgz
```

The Helm chart is located in the **jcnr** directory.

8. Modify the Helm chart `helmchart/jcnr/values.yaml` file to match the Helm chart configuration you saved earlier.

9. Install the Cloud-Native Router Helm chart.

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

```
helm install jcnr .
```

```
NAME: jcnr
LAST DEPLOYED: xxxxxxx
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

10. Check that the Cloud-Native Router Helm chart is being installed.

```
helm ls
```

Sample output:

```
NAME    NAMESPACE    REVISION      UPDATED    STATUS      CHART             APP VERSION
jcnr    default      1             xxxxxxx    deployed    jcnr-xxxxxxx      xxxxxxx
```

If the new version of Cloud-Native Router fails to install, troubleshoot the installation as you normally do. Look at the Cloud-Native Router deployer logs and see "Troubleshoot Deployment" on page 378.

11. Reconfigure cRPD with the saved Junos CLI commands.

    Access the cRPD CLI and use the Junos CLI `load` command to load the previously saved configuration.

Upgrade from Cloud-Native Router Release 24.2 and Later

1. Download the ***<sw_package>*.tar.gz** tarball to the directory of your choice. See "Cloud-Native Router Software Download Packages" on page 387 for the available package options.
2. Expand the downloaded package.

```
tar xzvf <sw_package>.tar.gz
```

3. Change directory to the main installation directory.

   If you're installing Cloud-Native Router only, then:

```
cd Juniper_Cloud_Native_Router_<release>
```

   This directory contains the Helm chart for Cloud-Native Router only.

   If you're installing Cloud-Native Router and cSRX at the same time, then:

```
cd Juniper_Cloud_Native_Router_CSRX_<release>
```

   This directory contains the combination Helm chart for both Cloud-Native Router and cSRX.

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

4. View the contents in the current directory.

```
ls
helmchart  images  README.md  secrets
```

5. Change to the **helmchart** directory and expand the Helm chart.

```
cd helmchart
```

```
ls
jcnr-<release>.tgz
```

```
tar -xzvf jcnr-<release>.tgz
```

```
ls
jcnr   jcnr-<release>.tgz
```

The Helm chart is located in the **jcnr** directory.

6. Modify the Helm chart `helmchart/jcnr/values.yaml` file to match the Helm chart configuration in your current installation.

7. Upgrade JCNR.

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

```
helm upgrade jcnr .
```

```
Release "jcnr" has been upgraded. Happy Helming!
NAME: jcnr
LAST DEPLOYED: xxxxxxx
NAMESPACE: default
STATUS: deployed
```

```
REVISION: 2
TEST SUITE: None
```

8. Check that the Cloud-Native Router Helm chart is being installed.

```
helm ls
```

Sample output:

```
NAME    NAMESPACE       REVISION        UPDATED         STATUS          CHART           APP
VERSION
jcnr    default         2               xxxxxxx         deployed        jcnr-xxxxxxx
xxxxxxx
```

If the new version of Cloud-Native Router fails to install, troubleshoot the installation as you normally do. Look at the Cloud-Native Router deployer logs and see "Troubleshoot Deployment" on page 378.

Downgrade/Rollback JCNR

To downgrade or roll back from the current version to an older or previous version, uninstall the current version and install the older or previous version.

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 deletes contents of Cloud-Native Router directories, deletes cRPD created interfaces and removes any Kubernetes objects created for JCNR. (See the `restoreInterfaces` attribute in the Helm chart.)

NOTE: Uninstalling Cloud-Native Router using Helm does not delete the `jcnr` namespace or the `jcnr-secrets`. Delete these manually if needed.

1. Uninstall JCNR.

```
helm uninstall jcnr
```

2. Wait for all Cloud-Native Router resources to be fully deleted before attempting reinstallation.

   Premature re-installation can lead to installation issues and may require manual steps for recovery. If this occurs, use one or more of the following commands to clean up the uninstallation:

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

Manage Cloud-Native Router Licenses

SUMMARY

Learn how to install and renew your Cloud-Native Router license.

IN THIS SECTION

A Cloud-Native Router license is required for you to use the containerized Routing Protocol Daemon (cRPD). Cloud-Native Router licensing is aligned with the 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 Cloud-Native Router software through your Juniper Networks representative.

For more information on JAL or for 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 from a pre-23.2 release to this release.

Installing Your License

Use this procedure to install your Cloud-Native Router license.

NOTE: You must obtain your license file from your account team and install it in the **jcnr-secrets.yaml** file as described in the procedures in this section. Without the proper base64-encoded license key and root password in the **jcnr-secrets.yaml** file, the cRPD pod may sometimes not enter Running state, but remain in CrashLoopBackOff state.

1. Encode your license in base64.

   ```
   base64 -w 0 licenseFile
   ```

   where licenseFile is the license file that you obtained from Juniper Networks.

   The output of this command is your base64-encoded license.

2. Copy and paste your base64-encoded license into **secrets/jcnr-secrets.yaml**.

   The **secrets/jcnr-secrets.yaml** file contains a parameter called `crpd-license`:

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

   If this is your first time adding your license, then replace `<add your license in base64 format>` with your base64-encoded license.

   If you're renewing your license, then replace your old base64-encoded license with your new base64-encoded license.

   Save and quit the file and continue with your installation.

Renewing Your License

Use this procedure to renew your Cloud-Native Router license.

When your Cloud-Native Router license expires, you'll receive a License Expired notification through syslog. Additionally, you can see the License Expired notification event in the Cloud-Native Router notification log file (typically **/var/log/jcnr/jcnr_notifications.json**). The notification looks something like this: `LICENSE_EXPIRED: License for feature Containerized routing protocol daemon with standard features(243) expired. Contact Juniper partner or account team.`

All Cloud-Native Router features continue to function even after your license expires but will cease to function the next time the cRPD pod restarts. To prevent this from occurring, contact your Juniper Networks representative as soon as possible to receive a new license.

When you receive your new license, follow these steps to renew your license in the current cluster:

1. Follow "Installing Your License" on page 353 to install your new license.
2. Apply your new license to the cluster.

   ```
   kubectl apply -f secrets/jcnr-secrets.yaml
   ```

3. Restart the cRPD pod(s) to pick up the new license.

   ```
   kubectl delete pod jcnr-xxx-crpd-0 -n jcnr
   ```

   When you delete a cRPD pod, a new one (with the new license) will be instantiated in its place. If you have more than one cRPD pod, remember to delete them all.
4. Verify that your license was installed properly.
   a. Access the cRPD pod.

      ```
      kubectl exec -it jcnr-xxx-crpd-0 -n jcnr -- bash
      ```

   b. Enter CLI mode and show the license.

      ```
      cli
      ```

      ```
      show system license
      ```

      The output should show that the `containerized-rpd-standard` license was installed.

      If the output shows that the license was not installed, then double check your steps or call Juniper Networks for support.

Allocate CPUs to the Cloud-Native Router Forwarding Plane

SUMMARY

Learn how to allocate CPU cores using static CPU allocation or using the Kubernetes CPU Manager.

IN THIS SECTION

The Cloud-Native Router installation Helm chart and the vRouter CRD provide you with a number of controls to allocate CPU cores to the Cloud-Native Router vRouter. You can specify the requested number of cores, the core limit, and the cores to be assigned, either through static CPU allocation or through the Kubernetes CPU Manager.

Allocate CPUs Using the Kubernetes CPU Manager

Use this procedure to allocate CPU cores to vRouter DPDK pods using the Kubernetes CPU Manager. This is the recommended approach if your cluster is running the Kubernetes CPU Manager.

1. Specify the resource limits and requests for the `contrail-vrouter-kernel-init-dpdk` and the `contrail_vrouter_agent_dpdk` containers.

   a. Locate the **helmchart/jcnr/charts/jcnr-vrouter/values.yaml** file in your installation directory.

   b. Edit that file to specify the resource limits and requests for both the `contrail-vrouter-kernel-init-dpdk` and the `contrail_vrouter_agent_dpdk` containers.

```
resources:
  limits:
    cpu: <number_of_cpus>
    memory: <memory>
  requests:
    cpu: <number_of_cpus>
    memory: <memory>
```

To guarantee that each container gets what it's asking for, set the same `cpu` value in both the `limits` and `requests` sections, and set the same `memory` value in both the `limits` and `requests` sections for each container.

2. Configure the Helm chart to specify the number of guaranteed vRouter CPUs that you want for the vRouter pods.

   In the main **values.yaml** file:

   a. Disable the static CPU allocation method of assigning CPU cores by commenting out the following lines:

   ```
   #cpu_core_mask: "2,3,22,23"
   #dpdkCtrlThreadMask: "2,3"
   #serviceCoreMask: "2,3"
   ```

   b. Configure the vRouter DPDK pods to use the guaranteed CPUs reserved by the Kubernetes CPU Manager.

   For example, to reserve 5 CPU cores:

   ```
   guaranteedVrouterCpus: 5
   ```

   This value must be:

   - greater or equal to the number of CPU cores configured for the `contrail-vrouter-kernel-init-dpdk` and the `contrail_vrouter_agent_dpdk` containers in **helmchart/charts/jcnr/jcnr-vrouter/values.yaml**, and
   - smaller or equal to the number of CPU cores reserved by the Kubernetes CPU Manager.

   The minimum recommended number is one more than the desired number of forwarding cores.

   c. Specify the number of CPU cores to use for vRouter DPDK service/control threads.

   For example, to reserve 1 core for vRouter DPDK service/control threads:

   ```
   numServiceCtrlThreadCPU: 1
   ```

   This leaves the remaining cores (four, in this example) for forwarding.

3. Proceed with your Cloud-Native Router installation.

4. After Cloud-Native Router is installed, check to make sure the vRouter DPDK pods has a QoS Class of Guaranteed.

```
kubectl get pod -n contrail contrail-vrouter-masters-vrdpdk-<xxxx> -o yaml | grep -i qosclass
```

The output should look like this:

```
qosClass: Guaranteed
```

5. To find out which CPUs are allocated to the vRouter DPDK container:

```
kubectl exec -n contrail contrail-vrouter-masters-vrdpdk-<xxxx> -c contrail-vrouter-agent-
dpdk -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

The output should list the cores assigned to the container.

6. To view the CPU assignment from the Kubernetes CPU Manager:

   a. SSH into a node where Cloud-Native Router is running.

   b. Look at the Kubernetes CPU Manager state.

      For example:

```
 cat /var/lib/kubelet/cpu_manager_state | jq
{
  "policyName": "static",
  "defaultCpuSet": "0-1,7-11",
  "entries": {
    "915d338f-c013-4984-a53c-51db78476dbf": {
      "contrail-vrouter-agent-dpdk": "2-6",
      "contrail-vrouter-kernel-init-dpdk": "2"
    }
  },
  "checksum": 3199431349
}
```

NOTE: You'll need to install `jq` (`dnf install -y jq`) in order to see formatted output.

Allocate CPUs Using Static CPU Allocation

Use this procedure to allocate CPU cores to vRouter DPDK pods using static CPU allocation. We recommend you use this method only when your cluster is not running the Kubernetes CPU Manager.

1. Specify the resource limits and requests for the `contrail-vrouter-kernel-init-dpdk` and the `contrail_vrouter_agent_dpdk` containers.

   a. Locate the **helmchart/jcnr/charts/jcnr-vrouter/values.yaml** file in your installation directory.

   b. Edit that file to specify the resource limits and requests for both the `contrail-vrouter-kernel-init-dpdk` and the `contrail_vrouter_agent_dpdk` containers.

   ```
   resources:
     limits:
       cpu: <number_of_cpus>
       memory: <memory>
     requests:
       cpu: <number_of_cpus>
       memory: <memory>
   ```

   To guarantee that each container gets what it's asking for, set the same `cpu` value in both the `limits` and `requests` sections, and set the same `memory` value in both the `limits` and `requests` sections for each container.

2. Configure the Helm chart to specify the cores that you want the vRouter DPDK to use.

   a. Disable the use of the Kubernetes CPU Manager for vRouter core allocation by commenting out the following:

   ```
   #guaranteedVrouterCpus: 5
   #numServiceCtrlThreadCPU: 1
   ```

   b. Specify the CPU cores to use for static CPU allocation.

   For example, to specify cores 2, 3, 22, and 23:

   ```
   cpu_core_mask: "2,3,22,23"
   ```

   c. Specify the CPU cores to use for vRouter DPDK service/control threads.

For example, to reserve cores 2 and 3 for vRouter DPDK service/control threads:

```
dpdkCtrlThreadMask: "2,3"
serviceCoreMask: "2,3"
```

This example leaves cores 22 and 23 for forwarding.

3. Proceed with your Cloud-Native Router installation.

Host Protection using Control Plane Policing

SUMMARY

This topic provides details about configuring Juniper Cloud-Native Router with host protection against DDoS attacks.

NOTE: This is a "Juniper Technology Preview" on page 452 feature.

Juniper Cloud-Native Router supports host protection against Distributed Denial of Service (DDoS) Attacks. You can configure rate-limiting for host traffic based on protocol classification on the loopback interface `lo0.0` using layer 3 class of service. See *Layer-3 Class of Service (CoS)* .

Here is a sample configlet to rate-limit BGP control plane traffic on the loopback `lo0.0` interface:

```
apiVersion: configplane.juniper.net/v1
kind: Configlet
metadata:
  name: configlet-sample
  namespace: jcnr
spec:
  config: |-
    set interfaces lo0.0 unit 0 family inet filter input f1
    set firewall three-color-policer test action loss-priority high then discard
    set firewall three-color-policer test two-rate color-blind
    set firewall three-color-policer test two-rate committed-information-rate 50m
```

```
    set firewall three-color-policer test two-rate committed-burst-size 70m
    set firewall three-color-policer test two-rate peak-information-rate 2048
    set firewall three-color-policer test two-rate peak-burst-size 2048
    set firewall family inet filter f1 term t1 from source-port bgp
    set firewall family inet filter f1 term t1 then accept
    set firewall family inet filter f1 term t1 then count c1
    set firewall family inet filter f1 term t1 then three-color-policer two-rate test
crpdSelector:
    matchLabels:
      node: worker
```

See *Layer-3 Class of Service (CoS)* for more details.

11

CHAPTER

Validate and Troubleshoot

IN THIS CHAPTER

Cloud-Native Router Readiness Checks

SUMMARY

Learn how Cloud-Native Router Readiness checks verify that your Cloud-Native Router installation is working properly.

Cloud-Native Router Readiness checks are a set of tests that validate your installation. They consist of preflight and postflight checks.

Preflight checks Preflight checks verify that your cluster nodes can support JCNR. The checks test for resource capacity, OS version, and other infrastructure requirements. Preflight checks run prior to Cloud-Native Router coming up.

Postflight checks Postflight checks verify that your Cloud-Native Router installation is working properly. The checks test for status and other basic functions. Postflight checks run after Cloud-Native Router comes up.

Cloud-Native Router Readiness is the name of the custom resource where these preflight and postflight checks are defined. When you enable Cloud-Native Router Readiness checks in **values.yaml** (`readinessCheck: true`), a set of pods that run these tests are created. For example:

```
kubectl get pods -A | grep preflight
```

```
cpuavailability-preflight-k8s-cp0-wtt92        0/1     Completed   0
cpuavailability-preflight-k8s-worker0-p7bxk    0/1     Completed   0
cpuinstructionset-preflight-k8s-cp0-4zmzq      0/1     Completed   0
cpuinstructionset-preflight-k8s-worker0-gh9zl  0/1     Completed   0
crpd-prerequisite-preflight-k8s-cp0-sqnjj      0/1     Completed   0
crpd-prerequisite-preflight-k8s-worker0-29psj  0/1     Completed   0
<trimmed>
```

NOTE: To see the list of checks that are currently performed, see "List of Cloud-Native Router Readiness Checks" on page 404.

When Cloud-Native Router Readiness checks are enabled, they run automatically when you install the Cloud-Native Router Helm chart. A failed check does not necessarily fail the installation. In some cases, the installation is allowed to continue.

To see the results of the checks, look at the results ConfigMap. For best viewing of the results, use the JSON processor jq.

NOTE: If jq is not installed on your OS by default, install it using the appropriate package manager for your OS. For example:

```
dnf install jq
```

```
apt install jq
```

Here's an example of the preflight results ConfigMap:

```
kubectl get cm preflight-results -n contrail-deploy -ojsonpath={".data.readinessStatus"} | jq
```

```
{
  "preflight": {
    "taskResults": [
      {
        "name": "cpuavailability",
        "taskExecutionTime": "2s",
        "message": "task completed (2/2)",
        "taskPassed": false,
        "failure_reason": "core 2 is not available on the node\ncore 2 is not available on the
node\n"
      },
<trimmed>
```

If there are any errors, fix them and then uninstall and reinstall JCNR.

Here's an example of the postflight results ConfigMap:

```
kubectl get cm postflight-results -n contrail-deploy -ojsonpath={".data.readinessStatus"} | jq
```


```
{
  "postflight": {
    "taskResults": [
      {
        "name": "jcnrresources",
        "taskExecutionTime": "0s",
        "message": "task completed (1/1)",
        "taskPassed": true
      }
    ],
    "result": "success"
  }
}
```


Although preflight and postflight checks can detect many common errors, some errors can slip through undetected. If the installation still fails after you fix all listed errors, look at the deployer and applier logs for more information:

- For deployer logs, see "Check Deployer Logs" on page 380.
- For applier logs, see "View Log Files" on page 384.

Also see the various general troubleshooting suggestions in "Troubleshoot Deployment" on page 378.

Validation Factory

SUMMARY

Validation Factory provides a framework to test and validate Juniper Cloud-Native Router deployments. It simplifies the evaluation of Cloud-Native Router solutions for customer adoption.

IN THIS SECTION

Overview

Validation Factory provides a library of well-defined test profiles that validate basic layer 3 features, including common sanity and performance tests for Cloud-Native Router deployments.

The aim of Validation Factory is to reduce the number of tests you need to manually create and execute when evaluating or qualifying a Cloud-Native Router deployment for a production environment.

Test execution is automated using a Kubernetes custom resource (CR) and the test result is published in a user-friendly format.

NOTE: Validation Factory is supported for Wind River deployments only.

Table 41 on page 365 shows the supported features.

Table 41: Validation Factory Supported Features

Supported Features	Description
Supported topology	• Cloud-Native Router deployed in two single-node clusters • Cloud-Native Router deployed in five single-node clusters We recommend that you validate with five single-node clusters.
Supported routing protocols	BGP, OSPF, IS-IS
Supported tests	MPLS-over-UDP, SR-MPLS, SRv6

Table 42 on page 366 shows the Validation Factory components.

Table 42: Components

Component	Description
Test Profile Library	A python-based, well-defined collection of test profiles for basic layer 3 Cloud-Native Router functionality. Each profile specifies: • Test-Description: A clear explanation of the functionality or behavior being tested. • Test Parameters: Configurable parameters to tailor the test profiles to specific scenarios. • Pass and Fail Criteria: Defined metrics or conditions that determine success or failure of the test case. The following test cases are executed: • End-to-end IPv4 & IPv6 traffic between pods. Both kernel & DPDK interfaces are included. • Restart routing process on cRPD. • Restart of pod running test traffic. • Respawn of the cRPD pod. • Respawn of the vRouter agent pod. • Respawn of the vRouter DPDK pod.
Kubernetes Operator	The framework is implemented as a Kubernetes custom resource (CR). The operator performs the following tasks: • Manages the test custom resource definitions (CRDs). • Parses the CR upon creation. • Orchestrates the test execution process by bringing up containerized test pods. • Monitors test pod execution and collects results. • Publishes test results in a user-friendly format.

Table 42: Components *(Continued)*

Component	Description
Test Pods	Test pods execute specific test profiles. The pods contain the necessary tools and libraries for network performance testing. The operator dynamically provisions the pods based on the requested test profile.
Results	The results are stored as a filesystem volume on the Kubernetes cluster.

Test Topology Manifest

The test topology manifest describes the part of your network that you want to test. Table 43 on page 367 shows the main configuration parameters.

Table 43: Main Test Topology Parameters

Parameter		Description
apiVersion		Set to `testtopology.validationfactory.juniper.net/v1`.
kind		Set to `TestTopology`.
metadata		
	name	The name you want to call this test topology manifest.
	namespace	Set to `jcnr`.
spec		

Table 43: Main Test Topology Parameters *(Continued)*

Parameter			Description
	global		
		platform	Set to `windriver`.
		auth	
		secret	The name of the secret that contains the kubeconfigs of all cluster nodes.
		crpd	
		username	Username to log in to cRPD via SSH.
		password	Password to log in to cRPD via SSH.
	cluster		An array of Cloud-Native Router clusters to be tested. We support the following: • two clusters, with each cluster consisting of a single (worker) node • five clusters, with each cluster consisting of a single (worker) node
		name	Name of the cluster.
		kubeconfigpath	Path to the kubeconfig file on a node in the specified cluster.
		nodes	An array of Cloud-Native Router nodes in the cluster.
		name	Name of the node.
		ip	IP address of the node.

Table 43: Main Test Topology Parameters *(Continued)*

Parameter				Description
		jumphost		(Optional) IP address of the jumphost to access the node.
	connections			An array of connections between the specified nodes as per the test topology. You're describing how your nodes are connected together. For example, if you have five nodes connected in a full mesh, then this array will contain ten connections.
		name		Name of the connection.
		node1		
			name	Name of the node at one end of the connection.
			interface	Name of the fabric interface on that node.
		node2		
			name	Name of the node at the other end of the connection.
			interface	Name of the fabric interface on that node.

Table 43: Main Test Topology Parameters *(Continued)*

Parameter				Description

		protocols		The underlay protocols running on this connection (link). Set to the same value for all connections.
				
				Valid values depend on the type of test you're running:
				
				• If the tunnel `type` is `mpls-over-udp`, then set to either `bgp`, `ospf`, or `isis`.
				
				• If the segment routing path `type` is `sr-mpls`, then set to both `isis` and `mpls`. For example:
				
				`protocols:`
				`  - isis`
				`  - mpls`
				
				• If the segment routing path `type` is `srv6`, then set to `isis`.

	tunnels[1]			An array of tunnels for MPLS-over-UDP. Omit this section if you're not testing MPLS-over-UDP.
		name		Name of the tunnel.
		type		Tunnel type. Set to `mpls-over-udp`.
		node1		
			name	Name of the node at one end of the tunnel.
		node2		
			name	Name of the node at the other end of the tunnel.

Table 43: Main Test Topology Parameters *(Continued)*

Parameter	Description
segmentroutings[1]	An array of segment routing paths for SR-MPLS and SRv6. Omit this section if you're not testing SR-MPLS or SRv6.
name	Name of the segment routing path.
type	Path type. Set to `sr-mpls` or `srv6`.

endpoints	The pair of endpoints for this path. For example:
	`endpoints:`
	`  - jcnr-node5`
	`  - jcnr-node6`

transit	An array of transit hops for SRv6 paths. For example:
	`transit:`
	`  - jcnr-node7`
	`  - jcnr-node8`

[1] Each test topology manifest is limited to only one type of test: `mpls-over-udp`, `sr-mpls`, or `srv6`.

- If you're testing MPLS-over-UDP, then include the `tunnels` section but omit the `segmentroutings` section.
- If you're testing SR-MPLS or SRv6, then include the `segmentroutings` section but omit the `tunnels` section.
- Within the `segmentroutings` section, the `type` must be the same for all paths. You cannot create a test that has a mix of SR-MPLS and SRv6 paths.

Here's a sample two-node test topology manifest:

```
apiVersion: testtopology.validationfactory.juniper.net/v1
kind: TestTopology
metadata:
  name: twonode-mplsoverudp
```

```
  namespace: jcnr
spec:
  global:
    platform: windriver
    auth:
      secret: sshauth
    crpd:
      username: root
      password: <password>
  cluster:
  - name: cluster1
    kubeconfigpath: /etc/kubernetes/admin.conf
    nodes:
    - name: jcnr-node5
      ip: 10.108.33.135
  - name: cluster2
    kubeconfigpath: /etc/kubernetes/admin.conf
    nodes:
    - name: jcnr-node6
      ip: 10.108.33.136
  connections:
  - name: toDC1
    node1:
      name: jcnr-node5
      interface: ens1f2
    node2:
      name: jcnr-node6
      interface: ens1f2
    protocols:
    - isis
  tunnels:
  - name: tun1
    type: mpls-over-udp
    node1:
      name: jcnr-node5
    node2:
      name: jcnr-node6
```

Here's a sample five-node test topology manifest:

```
apiVersion: testtopology.validationfactory.juniper.net/v1
kind: TestTopology
```

```
metadata:
  name: fiveonode-mplsoverudp
  namespace: jcnr
spec:
  global:
    platform: windriver
    auth:
      secret: sshauth
    crpd:
      username: root
      password: <password>
  cluster:
  - name: PE1
    kubeconfigpath: /etc/kubernetes/admin.conf
    nodes:
    - name: jcnr-node14
      ip: 10.204.8.14
  - name: PE2
    kubeconfigpath: /etc/kubernetes/admin.conf
    nodes:
    - name: jcnr-node16
      ip: 10.204.8.16
  - name: P1
    kubeconfigpath: /etc/kubernetes/admin.conf
    nodes:
    - name: jcnr-node13
      ip: 10.204.8.13
  - name: P2
    kubeconfigpath: /etc/kubernetes/admin.conf
    nodes:
    - name: jcnr-node12
      ip: 10.204.8.12
  - name: P3
    kubeconfigpath: /etc/kubernetes/admin.conf
    nodes:
    - name: jcnr-node15
      ip: 10.204.8.15


  connections:
  - name: PE1andP1
    node1:
      name: jcnr-node14
```

```
    interface: ens1f0np0
  node2:
    name: jcnr-node13
    interface: ens1f0
  protocols:
    - isis
- name: P1toPE2
  node1:
    name: jcnr-node13
    interface: ens1f1
  node2:
    name: jcnr-node16
    interface: ens2f0
  protocols:
    - isis
- name: PE1andP2
  node1:
    name: jcnr-node14
    interface: ens1f1np1
  node2:
    name: jcnr-node12
    interface: ens1f1
  protocols:
    - isis
- name: P2andP3
  node1:
    name: jcnr-node12
    interface: ens2f0
  node2:
    name: jcnr-node15
    interface: ens2f0
  protocols:
    - isis

- name: P3toPE2
  node1:
    name: jcnr-node15
    interface: ens2f1
  node2:
    name: jcnr-node16
    interface: ens2f1
  protocols:
    - isis
```

```
- name: P1toP2
  node1:
    name: jcnr-node13
    interface: ens1f2
  node2:
    name: jcnr-node12
    interface: ens1f0
  protocols:
    - isis
- name: P1toP3
  node1:
    name: jcnr-node13
    interface: ens1f3
  node2:
    name: jcnr-node15
    interface: ens1f0
  protocols:
    - isis


tunnels:
- name: tun1
  type: mpls-over-udp
  node1:
    name: jcnr-node14
  node2:
    name: jcnr-node16
```

Execute the Test Profiles

You can run the Validation Factory tests on any host that has access to the clusters you want to test. Typically, however, you would run the tests on the installation host in one of those clusters. The installation host is where you installed Helm and where you ran the Cloud-Native Router installation for that cluster.

All clusters must have Cloud-Native Router installed, and all nodes in all clusters must allow the same login credentials (username and password or SSH key).

1. Download the Validation Factory software package to the installation host in one of the clusters that you want to test. The installation host is where you have Helm installed and where you ran the Cloud-Native Router installation for that cluster.

You can download the Cloud-Native Router Validation Factory software package from the Juniper Networks software download site. See "Cloud-Native Router Software Download Packages" on page 387.

2. Gunzip and untar the software package.

```
tar -xzvf JCNR_Validation_Factory_<release>.tar.gz
```

3. Load the provided images on all nodes in the cluster. The images are located in the downloaded package.

   See "Deploy Prepackaged Images" on page 399.

4. If desired, configure the port where you want the test results to be accessible.

   Look for the following line in **validation-factory/values.yaml** and change the port number to your desired value:

```
nodePort: 30000
```

5. Install the Helm chart.

```
cd validation-factory
```

```
helm install validation-factory .
```

6. Create a secret with the login credentials and kubeconfigs of your clusters and apply it.

   valfac-secrets.yaml:

```
apiVersion: v1
kind: Secret
metadata:
   name: <name_of_secret>
   namespace: jcnr
type: Opaque
data:
   key: <base64-encoded_ssh_key>
   username: <base64-encoded_username>
   password: <base64-encoded_password>
   <cluster1_name>-kubeconfig: <base64-encoded_kubeconfig_of_cluster1>
   <cluster2_name>-kubeconfig: <base64-encoded_kubeconfig_of_cluster2>
```

```
<clusterN_name>-kubeconfig: <base64-encoded_kubeconfig_of_clusterN>
etc.
```

name	The name you want to call this secret.
key	The base64-encoded ssh key that allows `username` to log in to every node. If you specify the `key`, then you don't specify the `password`.
username	The base64-encoded username to log in to every node.
password	The base64-encoded password for `username` to log in to every node. If you specify the `password`, then you don't specify the `key`.
<cluster1_name>-kubeconfig	The base64-encoded kubeconfig file of the first cluster. The order that you list your clusters is not important, but *<cluster1_name>* must match the name of one of your clusters.
<cluster2_name>-kubeconfig	The base64-encoded kubeconfig file of the second cluster. *<cluster2_name>* must match the name of one of the remaining clusters.
<clusterN_name>-kubeconfig	The base64-encoded kubeconfig file of the Nth cluster. *<clusterN_name>* must match the name of one of the remaining clusters.

NOTE: You'll need to base64-encode most of the required information in the secrets manifest.

- To base64-encode a file: `base64 -w0 <file>`
- To base64-encode a string: `echo -n <string> | base64 -w0`

  Copy the output to the respective locations in the secrets manifest.

Apply the secret:

```
kubectl apply -f valfac-secrets.yaml
```

7. Configure the test topology manifest. See "Test Topology Manifest" on page 367.
8. Apply the manifest to begin test execution.

   For example:

```
kubectl create -f fivenode_topology_test.yaml
```

9. View the test results.

View the test results at **http:// *<cluster_node_IP>*:*<node_port>*/ *<test_topology_name>*/ *<test_topology_name>*.html**, where *<node_port>* is the value you set in step 4.

For example, if you executed the test profile named `fivenode-mplsoverudp` on a cluster with node IP address `10.0.0.100` and node port `30000`, you'll be able to see the results at **http://10.0.0.100:30000/fivenode-mplsoverudp/fivenode-mplsoverudp.html**.

Troubleshoot Deployment

SUMMARY

Learn how to troubleshoot your Cloud-Native Router deployment.

IN THIS SECTION

NOTE: We use the JSON processor jq to format the output of some commands. If jq is not installed on your OS by default, install it using the appropriate package manager for your OS. For example:

```
dnf install jq
```

```
apt install jq
```

Common Problems

Table 44 on page 379 lists some common deployment problems and remedies.

Table 44: Common Problems

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>
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 44: Common Problems *(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 *Access cRPD CLI* 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 logs for errors	See "Check Deployer Logs" on page 380.

Check Deployer Logs

The deployer logs should be one of the first places you look when you run into installation problems. To check the deployer logs:

a. List the deployer pod.

```
kubectl get pods -n contrail-deploy
```

Sample output:

```
NAME                                  READY   STATUS    RESTARTS   AGE
contrail-k8s-deployer-6fbf77bc7-5j67d  1/1    Running   0          24h
```

b. View the deployer logs.

```
kubectl logs contrail-k8s-deployer-6fbf77bc7-5j67d -n contrail-deploy
```

Verify vRouter and cRPD Health

1. Check the vRouter daemonset.

   a. List the daemonsets.

   ```
   kubectl get ds -n contrail
   ```

   Sample output:

   ```
   NAME                                  DESIRED   CURRENT   READY   UP-TO-DATE
   AVAILABLE   NODE SELECTOR   AGE
   jcnr-0-contrail-vrouter-nodes          1         1         1       1
   1           <none>          24h
   jcnr-0-contrail-vrouter-nodes-vrdpdk   1         1         1       1
   1           <none>          24h
   ```

   b. Get vRouter daemonset details.

   ```
   kubectl get ds jcnr-0-contrail-vrouter-nodes -n contrail -o json | jq
   ```

   Sample output:


   ```
   {
     "apiVersion": "apps/v1",
     "kind": "DaemonSet",
   ```

```
  "metadata": {
    "annotations": {
      "deprecated.daemonset.template.generation": "1"
    },
    "creationTimestamp": "2024-09-12T21:29:31Z",
    "generation": 1,
    "labels": {
      "app": "contrail-vrouter-nodes",
      "core.juniper.net/jcnrInstance": "jcnr-0",
      "core.juniper.net/jcnrInstanceNs": "jcnr",
      "core.juniper.net/nodeName": "k8s-worker0"
<trimmed>

  "status": {
    "currentNumberScheduled": 1,
    "desiredNumberScheduled": 1,
    "numberAvailable": 1,
    "numberMisscheduled": 0,
    "numberReady": 1,
    "observedGeneration": 1,
    "updatedNumberScheduled": 1
  }
}
```

2. Check the cRPD stateful set.

   a. List the stateful sets.

```
kubectl get sts -n jcnr
```

   Sample output:

```
NAME          READY   AGE
jcnr-0-crpd   1/1     24h
```

   b. Get the cRPD stateful set details.

```
kubectl get sts -n jcnr -o json | jq
```

Sample output:

```
{
  "apiVersion": "v1",
  "items": [
    {
      "apiVersion": "apps/v1",
      "kind": "StatefulSet",
      "metadata": {
        "creationTimestamp": "2024-09-12T21:29:23Z",
        "generation": 1,
        "labels": {
          "core.juniper.net/jcnrInstance": "jcnr-0",
          "core.juniper.net/jcnrInstanceNs": "jcnr"
        },
        "name": "jcnr-0-crpd",
        "namespace": "jcnr",
        "resourceVersion": "5502485",
        "uid": "025bc4b3-62eb-4552-9a83-b7e0123435d1"
      },
<trimmed>
```

Verify cRPD Configuration

The Cloud-Native Router deployment process creates a cRPD configuration file from the parameters in **values.yaml** for L2 mode and custom configuration via node annotations in L3 mode. This cRPD configuration file is at **/var/run/jcnr/juniper.conf** on any node running JCNR.

The cRPD configuration can be customized using node annotations. The cRPD pod will stay in `pending` state if the applied configuration is invalid.

The rendered custom configuration is in **/etc/crpd/juniper.conf.master**.

In an AWS EKS deployment you can see the rendered custom configuration by *accessing the cRPD CLI* and navigating to the **/config** directory.

View Log Files

You can find the Cloud-Native Router log files in the default `log_path` directory (**/var/log/jcnr/**) on any node running JCNR. You can change this location by changing the value of the `log_path` or `syslog_notifications` parameters in the **values.yaml** file prior to deployment.

Here's an example of some of the log files that Cloud-Native Router keeps.

```
ls /var/log/jcnr
```

```
applier
contrail-vrouter-agent.log
contrail-vrouter-dpdk-init.log
contrail-vrouter-dpdk.log
jcnr-cni.log
jcnr_notifications.json
license
messages
mgd-api
mosquitto
na-grpcd
vrouter-kernel-init.log
```

NOTE: If your deployment fails, check the applier logs in **applier/applier.log** for more information.

12

CHAPTER

Appendix

IN THIS CHAPTER

Kubernetes Overview

IN THIS SECTION

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).

Cloud-Native Router Software Download Packages

IN THIS SECTION

Cloud-Native Router Software Download Packages

Table 45 on page 388 shows the software packages available from the Juniper Networks software download site:

Table 45: Cloud-Native Router Software Download Packages

Package	Description
Juniper_Cloud_Native_Router_*<release>*.tar.gz	This contains the Helm chart for installing Cloud-Native Router on all deployments.
Juniper_Cloud_Native_Router_CSRX_*<release>*.tar.gz	This contains the combined Helm chart for installing Cloud-Native Router and cSRX on all deployments.
junos_csrx_*<release>*.tar.gz	This contains the Helm chart for installing cSRX on an existing Cloud-Native Router installation on all deployments.
Juniper_Cloud_Native_Router_Service_Module_*<release>*.tar.gz	This contains the Cloud-Native Router Operator Service Module Helm chart for the following: • installing the Cloud-Native Router VPC Gateway on an Amazon EKS deployment • installing the Cloud-Native Router host-based routing solution on a bare metal deployment
JCNR_Validation_Factory_*<release>*.tar.gz	This contains the Helm chart for installing Validation Factory.

NOTE: By default, the provided Helm charts download container images from the Juniper Networks `enterprise-hub.juniper.net` repository. Be sure to whitelist the https://enterprise-hub.juniper.net URL if you intend to use this default repository.

Cloud-Native Router Default Helm Chart

IN THIS SECTION

Default Helm Chart

This is the Cloud-Native Router release 24.4 default Helm chart **values.yaml** from the Juniper Networks Software Download site.

NOTE: This is not a working sample. Customize it for your deployment.

```
########################################################################
#                 Common Configuration (global vars)              #
########################################################################
global:

  # Set true/false to install syslog-ng.
  # It will deploy Service, ConfigMap and Daemonset about syslog-ng.
  installSyslog: true

  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: path/to/allimages/
  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: jcnr-container-prod/
      tag: 24.4.0.196
    crpd:
      repository: jcnr-container-prod/
      tag: 24.4R1.9
    jcnrcni:
      repository: jcnr-container-prod/
      tag: 24.4-20241112-741b53a
    telemetryExporter:
      repository: jcnr-container-prod/
      tag: 24.4.0.196
    tools:
      repository:
      tag: 24.4.0.196
    jcnrinit:
      repository: jcnr-container-prod/
      tag: 24.4.0.196
    readinessChecks:
      repository: jcnr-container-prod/
      tag: 24.4.0.196
    syslog:
      repository: jcnr-container-prod/
      tag: v6

  # Set true/false to Enable or Disable readiness checks (Pre / Post Flight tasks)
  # Pre-requisite - A configMap in the default namespace with the final deployer
manifests
  # Enable only for DPDK deployments
  readinessCheck: true

  # 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"

  #noLocalSwitching: [700]
  # Set AWS IAM Role for EKS PAYG deployments
  #iamrole: arn:aws:iam::298183613488:role/jcnr-payg-metering-role

  # 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
  #    qosSchedulerProfileName: "sched_profile_1"
  #- eth2:
  #    ddp: "off"                 # ddp parameter is optional; options include auto
```

```
or on or off; default: off
  #    qosSchedulerProfileName: "sched_profile_1"
  ########################

  # L2L3
  #- eth1:
  #    ddp: "auto"                # ddp parameter is optional; options include auto
or on or off; default: off
  #    qosSchedulerProfileName: "sched_profile_1"
  #- 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
  #  qosSchedulerProfileName: "sched_profile_1"
  #- 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
  #  qosSchedulerProfileName: "sched_profile_1"

  ##################################
  # 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"

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

  # path for the core file; vrouter considers /var/crash as default value
  #coreFilePath: /var/crash

  # 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 50053
  #grpcTelemetryPort: 50053

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default
port of 8081
  #vRouterDeployerPort: 8082


jcnr-vrouter:
  # do not configure cpu_core_mask if you wish to use Kubernetes CPU manager static
policy (pod with Guaranteed QoS) for vRouter DPDK
  # cpu_core_mask is the vRouter forward core mask i.e. if specified, vRouter will be
run using the mentioned cores
  cpu_core_mask: "2,3,22,23"

  # configure guaranteedVrouterCpus if you wish to use CPU manager static policy (pod
with Guaranteed QoS) for vRouter DPDK
  #guaranteedVrouterCpus: 4

  # configurable parameter for dpdk control threads
  #dpdkCtrlThreadMask: "2,3"

  # configurable parameter for service core mask
  #serviceCoreMask: "2,3"

  # no of cpus to be assigned to service and control threads if serviceCoreMask,
dpdkCtrlThreadMask and cpuCoreMask are not provided
  #numServiceCtrlThreadCPU: 1

  # Set no of cores to be used for schedulerLcores if CPU is not provided in the QOS
scheduler profile
  #numberOfSchedulerLcores: 2

  # 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"
  #    primaryInterface: "enp59s0f0v0"
  #    slaveNetworkDetails:               # This section only applies, when network
attachment definition is used as the input
  #    - name: srif0net0
  #      namespace: default

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

  # define the QoS scheduler profiles for fabric interfaces
  #qosSchedulerProfiles:
  #  sched_profile_1:
  #    bandwidth: 10 #Gbps
  #  sched_profile_2:
  #    cpu: 14
  #    bandwidth: 25 #Gbps

  # 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

  dpdkCommandAdditionalArgs: "--yield_option 0"

  # enable monitoring thread example:
  #  - logs appear every 100 seconds
```

```
  #  - log nl_counter & profile_histogram
  # loggingMask explanation:
  #  0b001 = nl_counter
  #  0b010 = lcore_timestamp
  #  0b100 = profile_histogram
  # dpdk_monitoring_thread_config:
  #  loggingMask: 5
  #  loggingInterval: 100

  # 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 TWAMP port for vrouter dpdk, allowed ports 862, 49152 - 65535
  #twampPort: 862

  # uio driver will be vfio-pci or uio_pci_generic. For azure, the driver is
uio_hv_generic
  vrouter_dpdk_uio_driver: "vfio-pci"

  # agentModeType will be dpdk or xdp. set agentModeType dpdk will bringup dpdk
datapath. set agentModeType to xdp to use ebpf.
  agentModeType: dpdk

  # fabricRpfCheckDisable: Set this flag to false to enable the RPF check on all the
fabric interfaces of the JNCR, by default RPF check is disabled
  #fabricRpfCheckDisable: false

  #telemetry:
  #  disable: false
  #  metricsPort: 8072
  #  logLevel: info         #Possible options: warn, warning, info, debug, trace, or
verbose
  #  gnmi:
  #    enable: true
  #    port: 8076
  #vrouter:
  #  telemetry:
  #    metricsPort: 8070
  #    logLevel: info       #Possible options: warn, warning, info, debug, trace, or
```

```
verbose
  #    gnmi:
  #      enable: true
  #      port: 8075
  # persistConfig: set this flag to true if you wish jcnr-operator generated pod
configuration to persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specied
  # to enable persist config
  #persistConfig: true

  # enableLocalPersistence: set this flag to true if you wish to persist the
configuration which is pushed through CLI or netConf.
  # enableLocalPersistence: true retains configurations locally, even after node
restart and JCNR upgrade
  #enableLocalPersistence: false

  ################### jcnr-operator/windriver section ###################
  # Interface bound type (0 - unbound interface, 1 - sriov pre-bound interface)
  # For WRCP deployment with pre-bound interface please set the field
(interfaceBoundType: 1)
  #interfaceBoundType: 1

  # NetworkDetails - list of network attachment definition
  #networkDetails:
  #  - ddp: "off"           # ddp parameter is optional; options include on or off;
default: off
  #    name: srif0net0      # network attachment definition name
  #    namespace: default   # namespace name where the network attachment definition
is created
  #  - ddp: "on"
  #    name: srif1net1
  #    namespace: default

  # NetworkDeviceResources
  #networkResources:
  #  limits:
  #    intel.com/pci_sriov_net_datanet0: "1"
  #    intel.com/pci_sriov_net_datanet1: "1"
  #  requests:
  #    intel.com/pci_sriov_net_datanet0: "1"
```

```
      #    intel.com/pci_sriov_net_datanet1: "1"
      #

    contrail-tools:
    #set it to true to install contrail-tools
      install: false
```

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 Cloud-Native Router Helm chart.

The Cloud-Native Router Helm chart uses your `enterprise-hub.juniper.net` credentials to pull images from the `enterprise-hub.juniper.net` repository.

The Cloud-Native Router Helm chart expects your credentials to be in a specific format. One way of ensuring your credentials are in the proper format is to use docker (podman).

1. Install docker if you don't already have docker installed.

   For example, for Rocky Linux:

   ```
   dnf install -y docker
   ```

2. Create a **.docker** directory. This is where you'll store our credentials.

   ```
   mkdir ~/.docker
   ```

3. Log in to the Juniper Networks `enterprise-hub.juniper.net` repository.

```
docker login enterprise-hub.juniper.net --authfile=/root/.docker/config.json
```

   Enter your `enterprise-hub.juniper.net` username and password when prompted. Your credentials are now stored in **~/.docker/config.json**.

4. 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
```

5. Navigate to the `Juniper_Cloud_Native_Router_`*`<release-number>`*`/helmchart/jcnr` directory. Replace the credentials placeholder in **values.yaml** with the encoded credentials.

   The **values.yaml** file has a `<base64-encoded-credential>` credentials placeholder. Simply replace the placeholder with the encoded credentials.

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

   Double check by searching for the encoded credentials in **values.yaml**.

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

   You should see the encoded credentials.

Deploy Prepackaged Images

Use this procedure to import Cloud-Native Router images to the container runtime from the downloaded Cloud-Native Router software packages.

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 packages.

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 where you want to run the downloaded software. Simply use the respective container runtime commands. We show you how to do this in the procedure below.

1. Locate the gzipped images tar file in the downloaded software package.
   - For regular Cloud-Native Router images, the gzipped images tar file is **Juniper_Cloud_Native_Router_*<release>*/images/jcnr-images.tar.gz**.
   - For Cloud-Native Router Service Module images, the gzipped images tar file is **Juniper_Cloud_Native_Router_Service_Module_*<release>*/images/jcnr-images.tar.gz**.
   - For Cloud-Native Router Validation Factory images, the gzipped images tar file is **JCNR_Validation_Factory_*<release>*/images/jcnr-valfac-images.tar.gz**.
2. Gunzip the images tar file.

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

   or

   ```
   gunzip jcnr-valfac-images.tar.gz
   ```
3. Copy the gunzipped images tar file to every node where you're installing the downloaded software.
4. SSH to one of the nodes and go to the directory where you copied the gunzipped images tar file.
5. Import the images to the container runtime.

   For regular Cloud-Native Router images:
   - containerd: `sudo ctr -n k8s.io images import jcnr-images.tar`
   - docker: `sudo docker load -i jcnr-images.tar`

   For Cloud-Native Router Service Module images:
   - podman: `sudo podman load -i jcnr-images.tar`

   For Cloud-Native Router Validation Factory images:
   - containerd: `sudo ctr -n k8s.io images import jcnr-valfac-images.tar`
   - docker: `sudo docker load -i jcnr-valfac-images.tar`
6. Check that the images have been imported.

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

7. Repeat steps 4 to 6 on each node where you're installing the downloaded software.

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.

Configure Huge Pages

SUMMARY

Learn how to configure huge pages for the Cloud-Native Router vRouter.

IN THIS SECTION

Huge pages make memory accesses more efficient by reducing the number of TLB (translation look-aside buffer) misses and are instrumental in getting the best performance from your Cloud-Native Router vRouter installation.

Configuring huge pages is a two-part procedure. First, specify the number and size of huge pages that you want the node to make available ("Configure the Number of Huge Pages Available on a Node" on page 401) and then configure the Cloud-Native Router vRouter to use these huge pages ("Configure the Number of Huge Pages to Use" on page 403).

By default, the Cloud-Native Router vRouter is already configured to use huge pages, so the second part is only necessary if you want to change the number of huge pages that you want the Cloud-Native Router vRouter to use.

Configure the Number of Huge Pages Available on a Node

Use this procedure to specify the number and size of huge pages that you want to make available on a node.

NOTE: This procedure does not apply to a Red Hat OpenShift or a Wind River Cloud Platform deployment.

1. Log in as root to the cluster node where you want to configure huge pages.
2. Configure GRUB to boot up the node with the desired number of huge pages.

   Add `GRUB_CMDLINE_LINUX_DEFAULT` values in **/etc/default/grub**.

   For example, the following configures the node to boot up with 10 x 1 GB huge pages (and SR-IOV pass-through support):

   ```
   GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G
   hugepages=10 intel_iommu=on iommu=pt"
   ```
3. Update GRUB.

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

   ```
   reboot
   ```
5. Log back in to the node.
6. Verify that the node has acquired the requested number of huge pages.

   Look at the output to confirm that GRUB has been configured to request the desired number of huge pages:

   ```
   cat /proc/cmdline
   ```

   Look at the output to confirm that the node has successfully acquired and made available the desired number of huge pages:

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

Configure the Number of Huge Pages to Use

By default, the Cloud-Native Router vRouter is already configured to use huge pages. Use this procedure to change the number of huge pages that the Cloud-Native Router vRouter uses.

1. On the host where you're running the Cloud-Native Router installation procedures, go to the Helm chart directory for the Cloud-Native Router vRouter.

```
cd helmchart/jcnr/charts/jcnr-vrouter
```

2. Configure the number of huge pages in the Cloud-Native Router vRouter **values.yaml** file.

   Change the below default `hugepages-1Gi` value in `limits` and `requests` to the number of your choice:

```
contrail_vrouter_agent_dpdk:
  image: contrail-vrouter-dpdk
  tag: *vrouter_tag
  pullPolicy: IfNotPresent
  resources:
    limits:
      cpu: 4
      memory: 1Gi
      hugepages-1Gi: 6Gi          # Hugepages must be enabled with default size as 1G;
minimum 6Gi to be used
    requests:
      cpu: 4
      memory: 1Gi
      hugepages-1Gi: 6Gi
```

   For example, to set huge pages to 10 GB:

```
hugepages-1Gi: 10Gi
```

   NOTE: The number you specify here must not be greater than the number of huge pages you've made available on the node.

3. Save and exit the file.

List of Cloud-Native Router Readiness Checks

SUMMARY

This section provides the list of Cloud-Native Router Readiness checks. This list may change from release to release.

IN THIS SECTION

Preflight and Postflight Checks

Table 46 on page 404 lists the hard preflight checks. A hard preflight check stops the installation if the check fails.

Table 46: Hard Preflight Checks

Preflight Check Name	Description
cpuavailability	Checks that the requested CPU cores are available.
cpuinstructionset	Checks that the processor supports the required instruction set.
crpdlicense	Checks for the presence of jcnr-secrets.
dpdk	Checks that the proper DPDK driver is installed.
hugepages	Checks that the required hugepages are available.
nads	Checks the network configuration when running with pre-bound interfaces in a Wind River deployment. This check is disabled in all other situations.
noderesources	Checks that the required CPU, memory, and storage resources are available.

Table 46: Hard Preflight Checks *(Continued)*

Preflight Check Name	Description
ports	Checks that there are no port conflicts.

Table 47 on page 405 lists the soft preflight checks. A soft preflight check does not stop the installation if the check fails.

Table 47: Soft Preflight Checks

Preflight Check Name	Description
crpd-prerequisites	Checks that the prerequisite modules for cRPD are installed.
k8sversion	Checks that the Kubernetes version is supported.
nic	Checks that the NIC is supported.
ntp	Checks that NTP is configured and the NTP server is reachable.
os	Checks that the OS version is supported.

Table 48 on page 405 lists the hard postflight checks. A hard postflight check stops the installation if the check fails.

Table 48: Hard Postflight Checks

Postflight Check Name	Description
jcnrresources	Checks the status of various Cloud-Native Router resources.

CloudFormation Template for EKS Cluster

You can use the CloudFormation template below to bring up an Amazon EKS cluster. This template creates a cluster that meets all the system requirements in "Minimum Host System Requirements for EKS" on page 149. Use it to quickly get a cluster up and running.

This template assumes you have a VPC and you have subnets associated with at least two availability zones (AZs).

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Amazon EKS Cluster with Node Group'

Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
      -
        Label:
          default: "EKS Configuration"
        Parameters:
          - ClusterName
          - ClusterVersion
          - NodeImageIdSSMParam
          - VpcId
          - SubnetIds
          - ExistingClusterSecurityGroups

      -
        Label:
          default: "NodeGroup Configuration"
        Parameters:
          - NodeGroupName
          - NodeInstanceType
          - NodeImageId
          - KeyName
          - ASGAutoAssignPublicIp
          - NodeAutoScalingGroupMinSize
          - NodeAutoScalingGroupDesiredSize
          - NodeAutoScalingGroupMaxSize
          - NodeVolumeSize
          - HugePageSize
```

```
          - ExistingNodeSecurityGroups
          - ExtraNodeSecurityGroups
          - ExtraNodeLabels


Parameters:
  ClusterName:
    Description: "Provide EKS cluster name for JCNR deployment. Ex: jcnr-payg-cloud-1"
    Type: String

  ClusterVersion:
    Description: Cluster Version
    Type: String
    Default: "1.28"
    AllowedValues:
      - "1.24"
      - "1.25"
      - "1.26"
      - "1.27"
      - "1.28"
      - "latest"

  VpcId:
    Description: "Provide VPC for JCNR EKS cluster"
    Type: AWS::EC2::VPC::Id

  SubnetIds:
    Description: Select minimum 2 subnets from each AvailabilityZones in above VPC
    Type: List<AWS::EC2::Subnet::Id>
    ConstraintDescription: Must be a list of at least two existing subnets associated with at
least two different availability zones. They should be residing in the selected Virtual Private
Cloud

  KeyName:
    Description: Key Pair to access Worker Nodes via SSH
    Type: AWS::EC2::KeyPair::KeyName

  NodeImageId:
    Type: String
    Default: ""
    Description: OPTIONAL - Only Specify AMI id for custom AMI to overwrite NodeImageIdSSMParam

  NodeImageIdSSMParam:
```

```
    Type: "AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>"
    Default: /aws/service/eks/optimized-ami/1.28/amazon-linux-2/recommended/image_id
    Description: "Match ClusterVersion in default value  Ex: If ClusterVersion is 1.27 , replace 
1.28 with 1.27"
    AllowedValues:
    - /aws/service/eks/optimized-ami/1.24/amazon-linux-2/recommended/image_id
    - /aws/service/eks/optimized-ami/1.25/amazon-linux-2/recommended/image_id
    - /aws/service/eks/optimized-ami/1.26/amazon-linux-2/recommended/image_id
    - /aws/service/eks/optimized-ami/1.27/amazon-linux-2/recommended/image_id
    - /aws/service/eks/optimized-ami/1.28/amazon-linux-2/recommended/image_id
    - /aws/service/eks/optimized-ami/latest/amazon-linux-2/recommended/image_id
    ConstraintDescription: Must matches with ClusterVersion parameter

  NodeInstanceType:
    Description: Worker Node Instance Type
    Type: String
    Default: m5.8xlarge
    ConstraintDescription: Must be a valid EC2 instance type

  NodeVolumeSize:
    Type: Number
    Description: Worker Node volume size
    Default: 30

  NodeAutoScalingGroupMinSize:
    Type: Number
    Description: Minimum size of Node Group ASG.
    Default: 1

  NodeAutoScalingGroupDesiredSize:
    Type: Number
    Description: Desired size of Node Group ASG.
    Default: 2

  NodeAutoScalingGroupMaxSize:
    Type: Number
    Description: Maximum size of Node Group ASG.
    Default: 2

  ASGAutoAssignPublicIp:
    Type: String
    Description: "auto assign public IP address for ASG instances"
    AllowedValues:
```

```
      - "yes"
      - "no"
    Default: "no"

  ExistingClusterSecurityGroups:
    Type: String
    Description: OPTIONAL - attach existing security group ID(s) for your nodegroup
    Default: ""

  ExtraNodeSecurityGroups:
    Type: String
    Description: OPTIONAL - attach extra existing security group ID(s) for your nodegroup
    Default: ""

  ExistingNodeSecurityGroups:
    Type: String
    Description: OPTIONAL - attach extra existing security group ID(s) for your nodegroup
    Default: ""

  ExtraNodeLabels:
    Description: Extra Node Labels(seperated by comma)
    Type: String
    Default: "jcnrcluster=cloud"

  NodeGroupName:
    Description: "Provide Worker Node group name. Ex: jcnr-nodegroup-1"
    Type: String

  HugePageSize:
    Type: Number
    Description: Huge Page size, minimum is 8GB
    Default: 8

Conditions:
  CreateLatestVersionCluster: !Equals [ !Ref ClusterVersion, latest ]
  CreateCustomVersionCluster: !Not [!Equals [!Ref ClusterVersion, latest]]
  HasNodeImageId: !Not [ !Equals [ !Ref NodeImageId, "" ] ]
  IsASGAutoAssignPublicIp: !Equals [ !Ref ASGAutoAssignPublicIp , "yes" ]
  AddExistingSG: !Not [ !Equals [ !Ref ExistingClusterSecurityGroups, "" ] ]
  CreateNewNodeSG: !Equals [ !Ref ExistingNodeSecurityGroups, "" ]
  AttachExistingNodeSG: !Not [ !Equals [ !Ref ExistingNodeSecurityGroups, "" ] ]
  AttachExtraNodeSG: !Not [ !Equals [ !Ref ExtraNodeSecurityGroups, "" ] ]
```

```
Rules:
  SubnetsInVPC:
    Assertions:
    - Assert:
        Fn::EachMemberIn:
        - Fn::ValueOfAll:
          - AWS::EC2::Subnet::Id
          - VpcId
        - Fn::RefAll: AWS::EC2::VPC::Id
      AssertDescription: All subnets must in the VPC

#
# Control Plane
#

Resources:
  EKSCluster:
    Type: "AWS::EKS::Cluster"
    Properties:
      Name: !Ref ClusterName
      ResourcesVpcConfig:
        SecurityGroupIds:
          !If
            - AddExistingSG
            - !Split [",", !Sub "${ControlPlaneSecurityGroup},${ExistingClusterSecurityGroups}"]
            -
              - !Ref ControlPlaneSecurityGroup
        SubnetIds: !Ref SubnetIds
      RoleArn: !GetAtt EksServiceRole.Arn
      AccessConfig:
        AuthenticationMode: "API_AND_CONFIG_MAP"
      Version:
        Fn::If:
          - CreateCustomVersionCluster
          - !Ref ClusterVersion
          - 1.28

  EksServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
```

```
      - Effect: "Allow"
        Principal:
          Service: "eks.amazonaws.com"
        Action: "sts:AssumeRole"
    Path: "/"
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
      - arn:aws:iam::aws:policy/AmazonEKSServicePolicy
    RoleName: !Sub "EksSvcRole-${ClusterName}"

ControlPlaneSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: Cluster communication with worker nodes
    VpcId: !Ref VpcId
    Tags:
      - Key: Name
        Value: !Sub "${ClusterName}-ControlPlaneSecurityGroup"

ControlPlaneIngressFromWorkerNodesHttps:
  Type: AWS::EC2::SecurityGroupIngress
  Properties:
    Description: Allow incoming HTTPS traffic (TCP/443) from worker nodes (for API server)
    GroupId: !Ref ControlPlaneSecurityGroup
    SourceSecurityGroupId: !Ref NodeSecurityGroup
    IpProtocol: tcp
    ToPort: 443
    FromPort: 443

ControlPlaneEgressToWorkerNodesKubelet:
  Type: AWS::EC2::SecurityGroupEgress
  Properties:
    Description: Allow outgoing kubelet traffic (TCP/10250) to worker nodes
    GroupId: !Ref ControlPlaneSecurityGroup
    DestinationSecurityGroupId: !Ref NodeSecurityGroup
    IpProtocol: tcp
    FromPort: 10250
    ToPort: 10250

ControlPlaneEgressToWorkerNodesHttps:
  Type: AWS::EC2::SecurityGroupEgress
  Properties:
    Description: Allow outgoing HTTPS traffic (TCP/442) to worker nodes (for pods running
```

```
extension API servers)
      GroupId: !Ref ControlPlaneSecurityGroup
      DestinationSecurityGroupId: !Ref NodeSecurityGroup
      IpProtocol: tcp
      FromPort: 443
      ToPort: 443

#
# Worker Nodes
#

  NodeSecurityGroup:
    Condition: CreateNewNodeSG
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Security group for all nodes in the cluster
      VpcId:
        !Ref VpcId
      Tags:
      - Key: !Sub "kubernetes.io/cluster/${ClusterName}"
        Value: "owned"
      - Key: Name
        Value: !Sub "${ClusterName}-cluster/NodeSecurityGroup"

  NodeSecurityGroupIngress:
    Condition: CreateNewNodeSG
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Allow node to communicate with each other
      GroupId: !Ref NodeSecurityGroup
      SourceSecurityGroupId: !Ref NodeSecurityGroup
      IpProtocol: '-1'

  NodeSecurityGroupFromControlPlaneIngress:
    Condition: CreateNewNodeSG
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Allow worker Kubelets and pods to receive communication from the cluster
control plane
      GroupId: !Ref NodeSecurityGroup
      SourceSecurityGroupId: !Ref ControlPlaneSecurityGroup
      IpProtocol: tcp
      FromPort: 10250
```

```
      ToPort: 10250

  NodeSecurityGroupFromControlPlaneOn443Ingress:
    Condition: CreateNewNodeSG
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Allow pods running extension API servers on port 443 to receive communication
from cluster control plane
      GroupId: !Ref NodeSecurityGroup
      SourceSecurityGroupId: !Ref ControlPlaneSecurityGroup
      IpProtocol: tcp
      FromPort: 443
      ToPort: 443

  NodeSecurityGroupFromSSHIngress:
    Condition: CreateNewNodeSG
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Allow ssh to worker nodes
      GroupId: !Ref NodeSecurityGroup
      IpProtocol: tcp
      FromPort: 22
      ToPort: 22
      CidrIp: 0.0.0.0/0

  NodeInstanceRole:
    DependsOn: EKSCluster
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
        - Effect: "Allow"
          Principal:
            Service: "ec2.amazonaws.com"
          Action: "sts:AssumeRole"
      Path: "/"
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
        - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
        - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
        - arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
        - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

```
TG:
  DependsOn: EKSCluster
  Type: "AWS::ElasticLoadBalancingV2::TargetGroup"
  Properties:
    HealthCheckIntervalSeconds: 15
    HealthCheckPath: /
    # HealthCheckPort: String
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 5
    HealthyThresholdCount: 2
    # Matcher: Matcher
    Name: !Sub "${ClusterName}"
    Port: 31742
    Protocol: HTTP
    TargetType: instance
    UnhealthyThresholdCount: 2
    VpcId: !Ref VpcId

NodeGroup:
  DependsOn: EKSCluster
  Type: "AWS::EKS::Nodegroup"
  Properties:
    UpdateConfig:
      MaxUnavailable: 1
    ScalingConfig:
      MinSize: !Ref NodeAutoScalingGroupMinSize
      DesiredSize: !Ref NodeAutoScalingGroupDesiredSize
      MaxSize: !Ref NodeAutoScalingGroupMaxSize
    Labels: {}
    Taints: []
    CapacityType: "ON_DEMAND"
    NodegroupName: !Ref NodeGroupName
    NodeRole: !GetAtt NodeInstanceRole.Arn
    Subnets: !Ref SubnetIds
    AmiType: "CUSTOM"
    LaunchTemplate:
      Version: !GetAtt MyLaunchTemplate.LatestVersionNumber
      Id: !Ref MyLaunchTemplate
    ClusterName: !Ref ClusterName
    InstanceTypes: []

CSIDriverAddon:
```

```
    DependsOn: EKSCluster
    Type: "AWS::EKS::Addon"
    Properties:
      AddonName: "aws-ebs-csi-driver"
      AddonVersion: "v1.28.0-eksbuild.1"
      ClusterName: !Ref ClusterName

  VPCCNIAddon:
    DependsOn: EKSCluster
    Type: "AWS::EKS::Addon"
    Properties:
      AddonName: "vpc-cni"
      AddonVersion: "v1.15.1-eksbuild.1"
      ClusterName: !Ref ClusterName

#
# Launch Template
#
  MyLaunchTemplate:
    Type: AWS::EC2::LaunchTemplate
    Properties:
      LaunchTemplateName: !Sub "eksLaunchTemplate-${AWS::StackName}"
      LaunchTemplateData:
        # SecurityGroupIds:
        #   - !Ref NodeSecurityGroup
        TagSpecifications:
          -
            ResourceType: instance
            Tags:
              - Key: ltname
                Value: !Sub "eksLaunchTemplate-${AWS::StackName}"
              - Key: "eks:cluster-name"
                Value: !Sub "${ClusterName}"
              - Key: !Sub "kubernetes.io/cluster/${ClusterName}"
                Value: "owned"
        UserData:
          Fn::Base64:
            !Sub |
            #!/bin/bash
            echo '#!/bin/bash
            modprobe vfio-pci
            modprobe vfio_iommu_type1
            modprobe allow_unsafe_interrupts=1
```

```
            modprobe 8021q
            echo Y > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode
            echo Y > /sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts
            cd /sys/module/vfio/parameters/
            echo Y > enable_unsafe_noiommu_mode
            exit 0' > /usr/local/bin/jcnr_startup
            chmod +x /usr/local/bin/jcnr_startup

            echo '[Unit]
            Description=/usr/local/bin/jcnr_startup Compatibility
            ConditionPathExists=/usr/local/bin/jcnr_startup

            [Service]
            Type=forking
            ExecStart=/usr/local/bin/jcnr_startup start
            TimeoutSec=0
            StandardOutput=tty
            RemainAfterExit=yes
            SysVStartPriority=99

            [Install]
            WantedBy=multi-user.target' > /etc/systemd/system/jcnr-startup.service
            sudo systemctl enable jcnr-startup
            sudo systemctl start jcnr-startup

            if [ ! -f /var/jcnr_startup_flag ]; then
              sudo sed -i 's/\(GRUB_CMDLINE_LINUX_DEFAULT=".*\)"/\1 default_hugepagesz=1G 
hugepagesz=1G hugepages=${HugePageSize} intel_iommu=on iommu=pt"/' /etc/default/grub
              grub2-mkconfig -o /boot/grub2/grub.cfg
              set -o xtrace
              /etc/eks/bootstrap.sh ${ClusterName}
              /opt/aws/bin/cfn-signal \
                          --exit-code $? \
                          --stack  ${AWS::StackName} \
                          --resource NodeGroup \
                          --region ${AWS::Region}
              touch /var/jcnr_startup_flag
              sleep 2m
              reboot
            fi
        KeyName: !Ref KeyName
        NetworkInterfaces:
          - DeviceIndex: 0
```

```
      AssociatePublicIpAddress:
        !If
          - IsASGAutoAssignPublicIp
          - 'true'
          - 'false'
      Groups:
        !If
          - CreateNewNodeSG
          - !If
              - AttachExtraNodeSG
              - !Split [",", !Sub "${NodeSecurityGroup},${ExtraNodeSecurityGroups}"]
              -
                - !Ref NodeSecurityGroup
          - !Split [",", !Ref ExistingNodeSecurityGroups ]
  ImageId:
    !If
      - HasNodeImageId
      - !Ref NodeImageId
      - !Ref NodeImageIdSSMParam
  InstanceType: !Ref NodeInstanceType
  BlockDeviceMappings:
    - DeviceName: /dev/xvda
      Ebs:
        VolumeSize: !Ref NodeVolumeSize
        VolumeType: gp2
        DeleteOnTermination: true
```

Cloud-Native Router Operator Service Module: Host-Based Routing Example Configuration Files

SUMMARY

This section contains example scripts and configuration files that you can use to create a service module host-based routing deployment.

IN THIS SECTION

Host-Based Routing: Example Scripts and Configuration Files to Install cRPD

IN THIS SECTION

Example cRPD Installation Script

The following example script installs cRPD on the node where you run the script. If cRPD is already running on the node, the script removes the running cRPD instance and installs a new instance. If the script finds an existing cRPD configuration file, it will reuse that configuration file. Otherwise, it will use the configuration file specified by the *CONFIG_TEMPLATE* variable that you set in the script.

Run this script with the proper *CONFIG_TEMPLATE* configuration file on every node in your cluster.

We provide sample *CONFIG_TEMPLATE* configuration files in "Example Control Plane Node Configuration File" on page 419 and "Example Worker Node Configuration File" on page 428.

install-crpd.sh:

```
set -o nounset
```

```
set -o errexit
SCRIPT_DIR=$(cd -P `dirname $0`; pwd)
NETWORK_NS="ns:/run/netns/crpd"
# Specify the config file. For example:
# ctl_plane_crpd_connectivity_template_5_node.conf or
worker_crpd_connectivity_template_5_node.conf
CONFIG_TEMPLATE=ctl_plane_crpd_connectivity_template_5_node.conf
POD_NAME=crpd
CONTAINER_NAME=crpd01
# Remove existing pod
## Stop all containers in pod crpd
POD_ID=$(sudo podman pod ls -fname=${POD_NAME} -q)
if [ -n "$POD_ID" ]; then
   sudo podman pod stop ${POD_ID}
   sudo podman pod rm ${POD_ID}
fi
# Create Pod in NS (Tested with podman 4.6.2)
#
sudo podman pod create --name ${POD_NAME} --network ${NETWORK_NS}
# create config dir
CRPD_CONFIG_DIR=/etc/crpd/config
sudo rm -rf ${CRPD_CONFIG_DIR}
sudo mkdir -p ${CRPD_CONFIG_DIR}
if [[ -f ${CRPD_CONFIG_DIR}/juniper.conf || -f ${CRPD_CONFIG_DIR}/juniper.conf.gz ]]; then
   echo "conf file exists"
else
   echo "initialize with base config"
   envsubst < ${CONFIG_TEMPLATE} > crpd_base_connectivity.conf
   sudo cp ${SCRIPT_DIR}/crpd_base_connectivity.conf ${CRPD_CONFIG_DIR}/juniper.conf
fi
sudo podman volume create crpd01-varlog --ignore
sudo podman run --rm -d --name ${CONTAINER_NAME} --pod ${POD_NAME} --privileged -v /etc/crpd/
config:/config:Z -v crpd01-varlog:/var/log -it enterprise-hub.juniper.net:/jcnr-container-prod/
crpd:24.4R1.9
# List
sudo podman pod ps --ctr-status --ctr-names --ctr-ids
```

Example Control Plane Node Configuration File

This configuration file is referenced by *CONFIG_TEMPLATE* in the cRPD installation script. There is one control plane node configuration file per control plane node. See Table 49 on page 425 through Table 51 on page 427 for the variable values to set for each control plane node.

ctl_plane_crpd_connectivity_template_5_node.conf:

```
groups {
   base {
      apply-flags omit;
      system {
         root-authentication {
            encrypted-password "<encrypted_password>"
         }
         commit {
            xpath;
            constraints {
               direct-access;
            }
            notification {
               configuration-diff-format xml;
            }
         }
         scripts {
            action {
               max-datasize 256m;
            }
            language python3;
         }
         services {
            netconf {
               ssh;
            }
            ssh {
               root-login allow;
               port 24;
            }
         }
         license {
            keys {
               key "<crpd-license-key>";
            }
         }
      }
   }
   connectivity {
```

```
    interfaces {
       lo0 {
          mtu 9216;
          unit 0 {
             family inet {
                address ${LO0_IP}/32;
             }
          }
       }
       veth-crpd {
          mtu 9216;
          unit 0 {
             family inet {
                address ${VETH_CRPD}/30;
             }
             # *** uncomment below if running dual stack ***
             #family inet6 {
             #   address ${VETH6_CRPD}/126;
             }
          }
       }
    }
    policy-options {
       policy-statement accept-podcidr {
          term accept {
             from {
                route-filter ${POD_CIDR} orlonger;
             }
             then accept;
          }
          then reject;
       }
       policy-statement export-direct {
          term 1 {
             from {
                route-filter ${LO0_IP_POOL} orlonger;
             }
             then accept;
          }
          then reject;
       }
       policy-statement export-evpn {
          term 1 {
```

```
            from protocol evpn;
            then accept;
         }
         then reject;
      }
      policy-statement export-veth {
         term 1 {
            from {
               protocol direct;
               route-filter ${VETH_PREFIX}/30 exact;
            }
            then accept;
         }
         term 2 {
            from protocol bgp;
            then accept;
         }
         then {

            # *** uncomment below if running dual stack ***
            #next policy;
            reject;
         }
      }
      # *** uncomment below if running dual stack ***
      #policy-statement export-veth-v6 {
      #    term 1 {
      #       from {
      #          protocol direct;
      #          route-filter ${VETH6_PREFIX}/126 exact;
      #       }
      #       then accept;
      #    }
      #    term 2 {
      #       from protocol bgp;
      #       then accept;
      #    }
      #    then reject;
      #}
   }
   routing-instances {
      master-calico-ri {
         instance-type vrf;
```

```
            protocols {
               bgp {
                  group calico-bgprtrgrp-master {
                     multihop;
                     local-address ${VETH_CRPD};
                     import accept-podcidr;
                     export export-evpn;
                     remove-private no-peer-loop-check;
                     peer-as 64512;
                     local-as 64600;
                     neighbor ${VETH_K8S};
                  }
                  # *** uncomment below if running dual stack ***
                  #group calico-bgprtrgrp-master6 {
                  #   multihop;
                  #   local-address ${VETH6_CRPD};
                  #   export export-evpn;
                  #   remove-private no-peer-loop-check;
                  #   peer-as 64512;
                  #   local-as 64600;
                  #   neighbor ${VETH6_K8S};
                  #}
               }
               evpn {
                  ip-prefix-routes {
                     advertise direct-nexthop;
                     encapsulation vxlan;
                     vni 4096;
                     # ***Include below line when running IPv4 only. Comment out if running dual 
stack.***
                     export export-veth;
                     # ***Include below line when running dual stack. Comment out if running 
IPv4 only.***
                     #export [ export-veth export-veth-v6 ];
                     route-attributes {
                        community {
                           import-action allow;
                           export-action allow;
                        }
                     }
                  }
               }
            }
```

```
            interface veth-crpd;
            vrf-target target:1:4;
        }
    }
    routing-options {
        route-distinguisher-id ${LO0_IP};
        router-id ${LO0_IP};
    }
    protocols {
        bgp {
            group crpd-master-bgprtrgrp {
                export export-direct;
                peer-as 64500;
                local-as 64500;
                neighbor ${MASTER1_PEER_ENS4_IP};
                neighbor ${MASTER2_PEER_ENS4_IP};
            }
            group crpd-worker-bgprtrgrp {
                multihop;
                export export-direct;
                peer-as 64500;
                local-as 64500;
                neighbor ${WORKER1_PEER_ENS4_IP};
                neighbor ${WORKER2_PEER_ENS4_IP};
            }
            group crpd-master-lo-bgprtrgrp {
                local-address ${LO0_IP};
                family evpn {
                    signaling;
                }
                peer-as 64600;
                local-as 64600;
                neighbor ${MASTER1_EVPN_PEER_IP};
                neighbor ${MASTER2_EVPN_PEER_IP};
            }
            group crpd-worker-lo-bgprtrgrp {
                local-address ${LO0_IP};
                family evpn {
                    signaling;
                }
                peer-as 64600;
                local-as 64600;
                neighbor ${WORKER1_EVPN_PEER_IP};
```

```
                neighbor ${WORKER2_EVPN_PEER_IP};
            }
            cluster ${LO0_IP};
        }
      }
    }
}
apply-groups base;
apply-groups connectivity;
```

Table 49: Node 1 (Control Plane Node) Example Settings

Variable	Setting
LO0_IP_POOL	10.12.0.0/24
LO0_IP	10.12.0.1
VETH_CRPD	10.1.1.2
VETH6_CRPD	2001:db8:1::2
VETH_PREFIX	10.1.1.0
VETH6_PREFIX	2001:db8:1::0
VETH_K8S	10.1.1.1
VETH6_K8S	2001:db8:1::1
POD_CIDR	192.168.0.0/24
MASTER1_EVPN_PEER_IP	10.12.0.2
MASTER2_EVPN_PEER_IP	10.12.0.3

Table 49: Node 1 (Control Plane Node) Example Settings *(Continued)*

Variable	Setting
WORKER1_EVPN_PEER_IP	10.12.0.4
WORKER2_EVPN_PEER_IP	10.12.0.5
MASTER1_PEER_ENS4_IP	192.168.1.102
MASTER2_PEER_ENS4_IP	192.168.1.103
WORKER1_PEER_ENS4_IP	192.168.1.104
WORKER2_PEER_ENS4_IP	192.168.1.105

Table 50: Node 2 (Control Plane Node) Example Settings

Variable	Setting
LO0_IP_POOL	10.12.0.0/24
LO0_IP	10.12.0.2
VETH_CRPD	10.1.2.2
VETH6_CRPD	2001:db8:2::2
VETH_PREFIX	10.1.2.0
VETH6_PREFIX	2001:db8:2::0
VETH_K8S	10.1.2.1
VETH6_K8S	2001:db8:2::1

Table 50: Node 2 (Control Plane Node) Example Settings *(Continued)*

Variable	Setting
POD_CIDR	192.168.0.0/24
MASTER1_EVPN_PEER_IP	10.12.0.1
MASTER2_EVPN_PEER_IP	10.12.0.3
WORKER1_EVPN_PEER_IP	10.12.0.4
WORKER2_EVPN_PEER_IP	10.12.0.5
MASTER1_PEER_ENS4_IP	192.168.1.1
MASTER2_PEER_ENS4_IP	192.168.1.3
WORKER1_PEER_ENS4_IP	192.168.1.4
WORKER2_PEER_ENS4_IP	192.168.1.5

Table 51: Node 3 (Control Plane Node) Example Settings

Variable	Setting
LO0_IP_POOL	10.12.0.0/24
LO0_IP	10.12.0.3
VETH_CRPD	10.1.3.2
VETH6_CRPD	2001:db8:3::2
VETH_PREFIX	10.1.3.0

Table 51: Node 3 (Control Plane Node) Example Settings *(Continued)*

Variable	Setting
VETH6_PREFIX	2001:db8:3::0
VETH_K8S	10.1.3.1
VETH6_K8S	2001:db8:3::1
POD_CIDR	192.168.0.0/24
MASTER1_EVPN_PEER_IP	10.12.0.1
MASTER2_EVPN_PEER_IP	10.12.0.2
WORKER1_EVPN_PEER_IP	10.12.0.4
WORKER2_EVPN_PEER_IP	10.12.0.5
MASTER1_PEER_ENS4_IP	192.168.1.1
MASTER2_PEER_ENS4_IP	192.168.1.2
WORKER1_PEER_ENS4_IP	192.168.1.4
WORKER2_PEER_ENS4_IP	192.168.1.5

Example Worker Node Configuration File

This configuration file is referenced by *CONFIG_TEMPLATE* in the cRPD installation script. There is one worker node configuration file per worker node. See Table 52 on page 433 and Table 53 on page 435 for the variable values to set for each worker node.

worker_crpd_connectivity_template_5_node.conf:

```
groups {
   base {
      apply-flags omit;
      system {
         root-authentication {
            encrypted-password "<encrypted_password>"
         }
         commit {
            xpath;
            constraints {
               direct-access;
            }
            notification {
               configuration-diff-format xml;
            }
         }
         scripts {
            action {
               max-datasize 256m;
            }
            language python3;
         }
         services {
            netconf {
               ssh;
            }
            ssh {
               root-login allow;
               port 24;
            }
         }
         license {
            keys {
               key "<crpd_license_key>";
            }
         }
      }
   }
   connectivity {
```

```
    interfaces {
       lo0 {
          mtu 9216;
          unit 0 {
             family inet {
                address ${LO0_IP}/32;
             }
          }
       }
       veth-crpd {
          mtu 9216;
          unit 0 {
             family inet {
                address ${VETH_CRPD}/30;
             }
             # *** uncomment below if running dual stack ***
             #family inet6 {
             #   address ${VETH6_CRPD}/126;
             #}
          }
       }
    }
    policy-options {
       policy-statement accept-podcidr {
          term accept {
             from {
                route-filter ${POD_CIDR} orlonger;
             }
             then accept;
          }
          then reject;
       }
       policy-statement export-direct {
          term 1 {
             from {
                route-filter ${LO0_IP_POOL} orlonger;
             }
             then accept;
          }
          then reject;
       }
       policy-statement export-evpn {
          term 1 {
```

```
                from protocol evpn;
                then accept;
            }
            then reject;
        }
        policy-statement export-veth {
            term 1 {
                from {
                    protocol direct;
                    route-filter ${VETH_PREFIX}/30 exact;
                }
                then accept;
            }
            term 2 {
                from protocol bgp;
                then accept;
            }
            then {
                # *** uncomment below if running dual stack ***
                #next policy;
                reject;
            }
        }
        # *** uncomment below if running dual stack ***
        #policy-statement export-veth-v6 {
        #   term 1 {
        #      from {
        #         protocol direct;
        #         route-filter ${VETH6_PREFIX}/126 exact;
        #      }
        #      then accept;
        #   }
        #   term 2 {
        #      from protocol bgp;
        #      then accept;
        #   }
        #   then reject;
        #}
    }
    routing-instances {
        worker-calico-ri {
            instance-type vrf;
            protocols {
```

```
                bgp {
                   group calico-bgprtrgrp-worker {
                      multihop;
                      local-address ${VETH_CRPD};
                      import accept-podcidr;
                      export export-evpn;
                      remove-private no-peer-loop-check;
                      peer-as 64512;
                      local-as 64600;
                      neighbor ${VETH_K8S};
                   }
                   # *** uncomment below if running dual stack ***
                   #group calico-bgprtrgrp-worker6 {
                   #    multihop;
                   #    local-address ${VETH6_CRPD};
                   #    export export-evpn;
                   #    remove-private no-peer-loop-check;
                   #    peer-as 64512;
                   #    local-as 64600;
                   #    neighbor ${VETH6_K8S};
                   #}
                }
                evpn {
                   ip-prefix-routes {
                      advertise direct-nexthop;
                      encapsulation vxlan;
                      vni 4300;
                      # ***Include below line when running IPv4 only. Comment out if running dual
stack.***
                      export export-veth;
                      # ***Include below line when running dual stack. Comment out if running
IPv4 only.***
                      #export [ export-veth export-veth-v6 ];
                      route-attributes {
                         community {
                            import-action allow;
                            export-action allow;
                         }
                      }
                   }
                }
             }
             interface veth-crpd;
```

```
            vrf-target target:1:4;
         }
      }
      routing-options {
         route-distinguisher-id ${LO0_IP};
         router-id ${LO0_IP};
      }
      protocols {
         bgp {
            group crpd-master-bgprtrgrp {
               multihop;
               export export-direct;
               peer-as 64500;
               local-as 64500;
               neighbor ${MASTER1_PEER_ENS4_IP};
               neighbor ${MASTER2_PEER_ENS4_IP};
               neighbor ${MASTER3_PEER_ENS4_IP};
            }
            group crpd-master-lo-bgprtrgrp {
               local-address ${LO0_IP};
               family evpn {
                  signaling;
               }
               peer-as 64600;
               local-as 64600;
               neighbor ${MASTER1_EVPN_PEER_IP};
               neighbor ${MASTER2_EVPN_PEER_IP};
               neighbor ${MASTER3_EVPN_PEER_IP};
            }
         }
      }
   }
}
apply-groups base;
apply-groups connectivity;
```

Table 52: Node 4 (Worker Node) Example Settings

Variable	Setting
LO0_IP_POOL	10.12.0.0/24

Table 52: Node 4 (Worker Node) Example Settings *(Continued)*

Variable	Setting
LO0_IP	10.12.0.4
VETH_CRPD	10.1.4.2
VETH6_CRPD	2001:db8:4::2
VETH_PREFIX	10.1.4.0
VETH6_PREFIX	2001:db8:4::0
VETH_K8S	10.1.4.1
VETH6_K8S	2001:db8:4::1
POD_CIDR	192.168.0.0/24
MASTER1_EVPN_PEER_IP	10.12.0.1
MASTER2_EVPN_PEER_IP	10.12.0.2
MASTER3_EVPN_PEER_IP	10.12.0.3
MASTER1_PEER_ENS4_IP	192.168.1.101
MASTER2_PEER_ENS4_IP	192.168.1.102
MASTER3_PEER_ENS4_IP	192.168.1.103

Table 53: Node 5 (Worker Node) Example Settings

Variable	Setting
LO0_IP_POOL	10.12.0.0/24
LO0_IP	10.12.0.5
VETH_CRPD	10.1.5.2
VETH6_CRPD	2001:db8:5::2
VETH_PREFIX	10.1.5.0
VETH6_PREFIX	2001:db8:5::0
VETH_K8S	10.1.5.1
VETH6_K8S	2001:db8:5::1
POD_CIDR	192.168.0.0/24
MASTER1_EVPN_PEER_IP	10.12.0.1
MASTER2_EVPN_PEER_IP	10.12.0.2
MASTER3_EVPN_PEER_IP	10.12.0.3
MASTER1_PEER_ENS4_IP	192.168.1.101
MASTER2_PEER_ENS4_IP	192.168.1.102
MASTER3_PEER_ENS4_IP	192.168.1.103

Host-Based Routing: Example Calico Configuration

IN THIS SECTION

BGP Configuration Example

bgpconfig.yaml:

```
apiVersion: crd.projectcalico.org/v1
kind: BGPConfiguration
metadata:
   name: default
spec:
   asNumber: 64512
   listenPort: 1179
   logSeverityScreen: Debug
   nodeToNodeMeshEnabled: false
```

IP Pool Configuration Example

ippool-v4.yaml:

```
apiVersion: crd.projectcalico.org/v1
kind: IPPool
metadata:
   name: default-ipv4-ippool
   spec:
   allowedUses:
   - Workload
   blockSize: 26
```

```
   cidr: 192.168.7.0/24
   ipipMode: Never
   natOutgoing: true
   nodeSelector: all()
   vxlanMode: Never
```

ippool-v6.yaml:

```
apiVersion: crd.projectcalico.org/v1
kind: IPPool
metadata:
   name: default-ipv6-ippool
   spec:
   allowedUses:
   - Workload
   blockSize: 122
   cidr: 2001:db8:42:0::/56
   ipipMode: Never
   natOutgoing: true
   nodeSelector: all()
   vxlanMode: Never
```

BGP Peer Configuration Example

bgppeers-v4.yaml:

```
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
metadata:
   name: node1
spec:
   sourceAddress: None
   asNumber: 64600
   node: node1
   peerIP: 10.1.1.2:179
---
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
```

```
metadata:
   name: node2
spec:
   sourceAddress: None
   asNumber: 64600
   node: node2
   peerIP: 10.1.2.2:179
---
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
metadata:
   name: node3
spec:
   sourceAddress: None
   asNumber: 64600
   node: node3
   peerIP: 10.1.3.2:179
---
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
metadata:
   name: node4
spec:
   sourceAddress: None
   asNumber: 64600
   node: node4
   peerIP: 10.1.4.2:179
---
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
metadata:
   name: node5
spec:
   sourceAddress: None
   asNumber: 64600
   node: node5
   peerIP: 10.1.5.2:179
```

bgppeers-v6.yaml:

```
apiVersion: crd.projectcalico.org/v1
```

```
kind: BGPPeer
metadata:
# Change for every node
   name: node1-ipv6
spec:
   sourceAddress: None
   asNumber: 64600
   node: node1
   peerIP: '[2001:db8:1::2]:179'
---
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
metadata:
# Change for every node
   name: node2-ipv6
spec:
   sourceAddress: None
   asNumber: 64600
   node: node2
   peerIP: '[2001:db8:2::2]:179'
---
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
metadata:
# Change for every node
   name: node3-ipv6
spec:
   sourceAddress: None
   asNumber: 64600
   node: node3
   peerIP: '[2001:db8:3::2]:179'
---
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
metadata:
# Change for every node
   name: node4-ipv6
spec:
   sourceAddress: None
   asNumber: 64600
   node: node4
   peerIP: '[2001:db8:4::2]:179'
---
```

```
apiVersion: crd.projectcalico.org/v1
kind: BGPPeer
metadata:
# Change for every node
   name: node5-ipv6
spec:
   sourceAddress: None
   asNumber: 64600
   node: node5
   peerIP: '[2001:db8:5::2]:179'
```

Host-Based Routing: Example VxLAN and Route Target Pools

IN THIS SECTION

VxLAN Pool Example

vxlan-pool.yaml:

```
apiVersion: core.svcmodule.juniper.net/v1
kind: Pool
metadata:
   name: default-vni
   namespace: svcmodule-system
spec:
   vxlanId:
      start: 4096
      end: 16777215
```

Route Target Pool Example

rt-pool.yaml:

```
apiVersion: core.svcmodule.juniper.net/v1
kind: Pool
metadata:
   name: default-route-target-number
   namespace: svcmodule-system
spec:
   routeTarget:
      start: 8000000
      size: 2048
```

Host-Based Routing: Example JCNR Configuration

IN THIS SECTION

JCNR Configuration

jcnr-config.yaml:

```
apiVersion: configplane.juniper.net/v1
kind: Jcnr
metadata:
   name: crpd-master
   namespace: hbn
spec:
   replicas: 3
   jcnrTemplate:
      externallyInitialized: true
      loopbackAddressInitialized: true
```

```
      nodeSelector:
        master: ""
---
apiVersion: configplane.juniper.net/v1
kind: Jcnr
metadata:
  name: crpd-worker
  namespace: hbn
spec:
  replicas: 2
  jcnrTemplate:
    externallyInitialized: true
    loopbackAddressInitialized: true
    nodeSelector:
      worker: ""
```

Host-Based Routing: Example Secondary CNI Configuration Files

IN THIS SECTION

Example MACVLAN Custom Resource

macvlan-cr.yaml:

```
apiVersion: core.svcmodule.juniper.net/v1
kind: RoutingInstance
metadata:
  name: macvlan-ri-master
  namespace: hbn
spec:
```

```
  crpdGroupReference:
    name: crpd-master
  instanceType: mac-vrf
  vrfTarget:
    importExport:
      name: target:64512:8000000
  routingOptions:
    routeDistinguisherId: 192.168.100.2:11
  bridgeDomains:
  - name: test-domain
    interface: vrf-end
    vLanId: 100
    vni: 4200
---
apiVersion: core.svcmodule.juniper.net/v1
kind: RoutingInstance
metadata:
  name: macvlan-ri-worker
  namespace: hbn
spec:
  crpdGroupReference:
    name: crpd-worker
  instanceType: mac-vrf
  vrfTarget:
    importExport:
      name: target:64512:8000000
  routingOptions:
    routeDistinguisherId: 192.168.100.2:11
  bridgeDomains:
  - name: test-domain
    interface: vrf-end
    vLanId: 100
    vni: 4200
---
apiVersion: core.svcmodule.juniper.net/v1
kind: EVPN
metadata:
  name: macvlan-evpn-master
  namespace: hbn
spec:
  encapsulation: vxlan
  defaultGateway: no-gateway-community
  routingInstanceParent:
```

```
    name: macvlan-ri-master
---
apiVersion: core.svcmodule.juniper.net/v1
kind: EVPN
metadata:
  name: macvlan-evpn-worker
  namespace: hbn
spec:
  encapsulation: vxlan
  defaultGateway: no-gateway-community
  routingInstanceParent:
    name: macvlan-ri-worker
---
apiVersion: core.svcmodule.juniper.net/v1
kind: InterfaceGroup
metadata:
  name: jcnr-macvlan-master
  namespace: hbn
spec:
  instanceParent:
    parentType: jcnr
  reference:
    name: crpd-master
interfaceName: vrf-end
interfaceTemplate:
  encapsulation: vlan-bridge
  families:
    - addressFamily: bridge
---
apiVersion: core.svcmodule.juniper.net/v1
kind: InterfaceGroup
metadata:
  name: jcnr-macvlan-worker
  namespace: hbn
spec:
  instanceParent:
    parentType: jcnr
    reference:
      name: crpd-worker
  interfaceName: vrf-end
  interfaceTemplate:
    encapsulation: vlan-bridge
```

```
    families:
      - addressFamily: bridge
```

Example MACVLAN Pods

macvlan-pods.yaml:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan-conf
spec:
  config: '{
      "cniVersion": "0.3.1",
      "plugins": [
        {
          "type": "macvlan",
          "capabilities": { "ips": true },
          "master": "host-end",
          "mode": "bridge",
          "ipam": {
            "type": "static",
            "routes": [
              {
                "dst": "0.0.0.0/0",
                "gw": "10.9.1.1"
              }
            ]
          }
        }, {
          "capabilities": { "mac": true },
          "type": "tuning"
        }
      ]
    }'
---
apiVersion: v1
kind: Pod
metadata:
  name: l2-pod-1
```

```
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
            { "name": "macvlan-conf",
              "ips": [ "10.9.1.101/24" ],
              "mac": "00:53:57:49:47:aa",
              "gateway": [ "10.9.1.1" ]
            }]'
spec:
  containers:
  - name: l2-pod-1
    command: ["/bin/bash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: google-containers/toolbox
    ports:
    - containerPort: 80
    securityContext:
      capabilities:
        add:
        - NET_ADMIN
      privileged: true
  automountServiceAccountToken: false
  nodeName: ${node-name}
---
apiVersion: v1
kind: Pod
metadata:
  name: l2-pod-2
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
            { "name": "macvlan-conf",
              "ips": [ "10.9.1.102/24" ],
              "mac": "00:53:57:49:47:bb",
              "gateway": [ "10.9.1.1" ]
            }]'
spec:
  containers:
  - name: samplepod
    command: ["/bin/bash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: google-containers/toolbox
    ports:
    - containerPort: 80
    securityContext:
      capabilities:
        add:
```

```
        - NET_ADMIN
      privileged: true
  automountServiceAccountToken: false
  nodeName: ${node-name}
```

Example IPVLAN Custom Resource

ipvlan-cr.yaml:

```
apiVersion: core.svcmodule.juniper.net/v1
kind: RoutingPolicy
metadata:
  name: static-rt
  namespace: hbn
spec:
  terms:
    - name: learned-from-static
      from:
        protocol: static
      then:
        accept: true
  default:
    accept: false
---
apiVersion: core.svcmodule.juniper.net/v1
kind: RoutingInstance
metadata:
  name: ipvlan-ri-master
  namespace: hbn
spec:
  crpdGroupReference:
    name: crpd-master
  instanceType: vrf
  interfaces:
  - ipvlan-vrf
  vrfTarget:
    importExport:
      name: target:11:11
  routingOptions:
    routeDistinguisherId: 11:11
---
```

```
apiVersion: core.svcmodule.juniper.net/v1
kind: RoutingInstance
metadata:
  name: ipvlan-ri-worker
  namespace: hbn
spec:
  crpdGroupReference:
    name: crpd-worker
  instanceType: vrf
  interfaces:
  - ipvlan-vrf
  vrfTarget:
    importExport:
      name: target:11:11
  routingOptions:
    routeDistinguisherId: 11:11
---
apiVersion: core.svcmodule.juniper.net/v1
kind: EVPN
metadata:
  name: ipvlan-evpn-master
  namespace: hbn
spec:
  encapsulation: vxlan
  exportPolicy:
    name: static-rt
  routingInstanceParent:
    name: ipvlan-ri-master
---
apiVersion: core.svcmodule.juniper.net/v1
kind: EVPN
metadata:
  name: ipvlan-evpn-worker
  namespace: hbn
spec:
  encapsulation: vxlan
  exportPolicy:
    name: static-rt
  routingInstanceParent:
    name: ipvlan-ri-worker
---
apiVersion: core.svcmodule.juniper.net/v1
kind: InterfaceGroup
```

```
metadata:
  name: jcnr-ipvlan-master
  namespace: hbn
spec:
  instanceParent:
    parentType: jcnr
    reference:
      name: crpd-master
  interfaceName: ipvlan-vrf
  interfaceTemplate:
    families:
      - addressFamily: inet
        ipAddress: 10.19.19.1/24
---
apiVersion: core.svcmodule.juniper.net/v1
kind: InterfaceGroup
metadata:
  name: jcnr-ipvlan-worker
  namespace: hbn
spec:
  instanceParent:
    parentType: jcnr
    reference:
      name: crpd-worker
  interfaceName: ipvlan-vrf
  interfaceTemplate:
    families:
      - addressFamily: inet
        ipAddress: 10.19.19.1/24
---
apiVersion: configplane.juniper.net/v1
kind: NodeConfiglet
metadata:
  labels:
    core.juniper.net/nodeName: <node-name where ipvlan-pod-1 will be scheduled>
  name: ipvlan-addon-node-1
  namespace: hbn
spec:
  clis:
  - set routing-instances <name of RI to which node belongs to> routing-options static route
10.19.19.101/32 nexthop 10.19.19.101
  nodeName: <node-name where ipvlan-pod-1 will be scheduled>
---
```

```
apiVersion: configplane.juniper.net/v1
kind: NodeConfiglet
metadata:
  labels:
    core.juniper.net/nodeName: <node-name where ipvlan-pod-2 will be scheduled>
  name: ipvlan-addon-node-2
  namespace: hbn
spec:
  clis:
  - set routing-instances <name of RI to which node belongs to> routing-options static route
10.19.19.102/32 nexthop 10.19.19.102
  nodeName: <node-name where ipvlan-pod-2 will be scheduled>
```

Example IPVLAN Pods

ipvlan-pods.yaml:

```


apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: ipvlan-conf
spec:
  config: '{
      "cniVersion": "0.3.1",
      "name": "ipvlan-conf",
      "type": "ipvlan",
      "master": "ipvlan-host",
      "mode": "l2",
      "ipam": {
          "type": "static"
      }
    }'
---
apiVersion: v1
kind: Pod
metadata:
  name: ipvlan-pod-1
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
            { "name": "ipvlan-conf",
```

```
            "ips": [ "10.19.19.101/24" ]
          }]'
spec:
  containers:
  - name: samplepod-1
    command: ["/bin/bash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: google-containers/toolbox
    ports:
    - containerPort: 80
    securityContext:
      capabilities:
        add:
        - NET_ADMIN
      privileged: true
  automountServiceAccountToken: false
  nodeName: ${node-name}
---
apiVersion: v1
kind: Pod
metadata:
  name: ipvlan-pod-2
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
          { "name": "ipvlan-conf",
            "ips": [ "10.19.19.102/24" ]
          }]'
spec:
  containers:
  - name: samplepod
    command: ["/bin/bash", "-c", "trap : TERM INT; sleep infinity & wait"]
    image: google-containers/toolbox
    ports:
    - containerPort: 80
    securityContext:
      capabilities:
        add:
        - NET_ADMIN
      privileged: true
  automountServiceAccountToken: false
  nodeName: ${node-name}
```

Juniper Technology Preview

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.