JUNIPER NETWORKS | Engineering Simplicity

Juniper Cloud-Native Router Deployment Guide

Published
2024-01-08

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

Juniper Cloud-Native Router Deployment Guide

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

YEAR 2000 NOTICE

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

END USER LICENSE AGREEMENT

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

Table of Contents

CHAPTER 1

Introduction

Juniper Cloud-Native Router Overview

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 Contrail® Data Plane Development Kit (DPDK) vRouter forwarding plane. It is implemented in Kubernetes and interacts seemlessly with a Kubernetes container network (CNI) framework.

Use Cases

The Cloud-Native Router has the following use cases:

- **Radio Access Network (RAN)**

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

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

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

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

Architecture and Key Components

The Juniper Cloud-Native Router consists of the Junos containerized routing protocol Daemon (cRPD) as the control plane (JCNR Controller), providing topology discovery, route advertisement and forwarding information base (FIB) programming, as well as dynamic underlays and overlays. It uses the Data Plane Development Kit (DPDK) enabled vRouter as a forwarding plane, providing packet forwarding for DPDK applications in a pod and host path I/O for protocol sessions. The third component is the JCNR 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.

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

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

Features

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

Juniper Cloud-Native Router Components

SUMMARY

The Juniper Cloud-Native Router solution consists of several components including the JCNR controller, JCNR vRouter and the JCNR-CNI. This topic provides a brief overview of the components of the Juniper Cloud-Native Router.

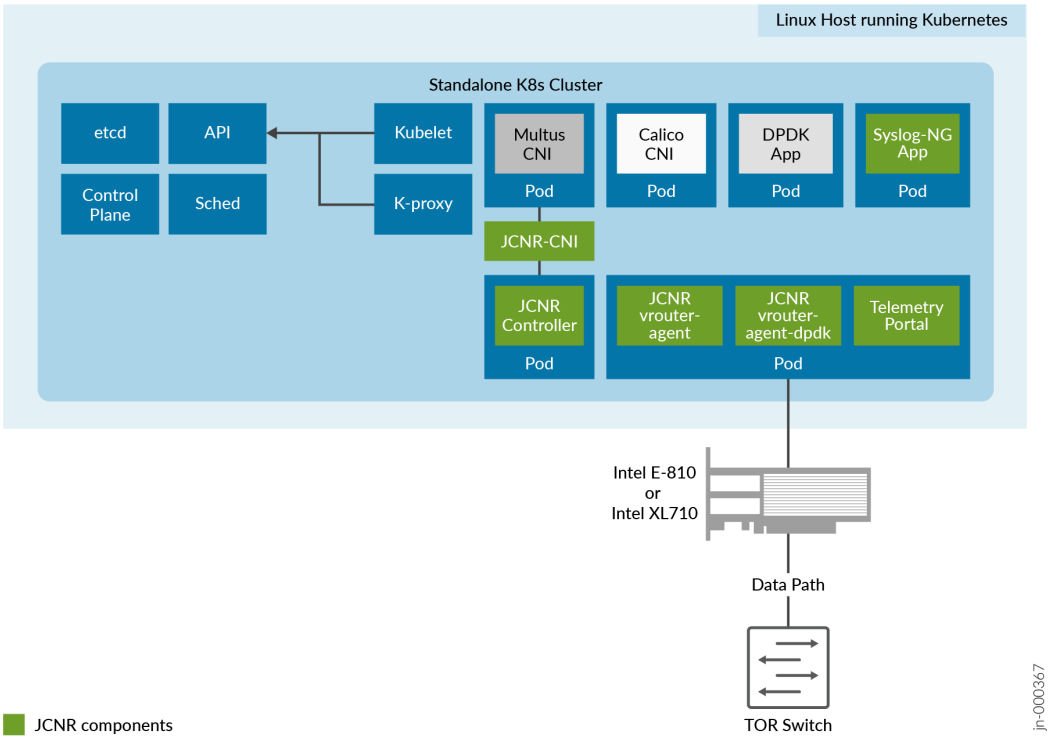
IN THIS SECTION

JCNR Components

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

The Figure 1 on page 6 shows the components of the Juniper Cloud-Native Router inside a Kubernetes cluster

Figure 1: Components of Juniper Cloud-Native Router



JCNR Controller

The JCNR 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 other components, primarily the JCNR vRouter, for implementation.

For example, firewall filters (ACLs) are supported on the controller to configure L2 access lists with deny rules. The controller sends the configuration information to the JCNR 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 from Release 23.2, establishing peer relationships with other BGP speakers to exchange routing information.

Configuration Options

During deployment, you can "Customize JCNR Configuration " on page 54 .

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

JCNR vRouter

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

JCNR vRouter Functionality:

- Performs routing with Layer 3 virtual private networks.
- Performs L2 forwarding.
- Supports high-performance DPDK-based forwarding.

Benefits of vRouter:

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

JCNR-CNI

JCNR-CNI is a new container network interface (CNI) developed by Juniper. JCNR-CNI is a Kubernetes CNI plugin installed on each node to provision network interfaces for application pods. During pod creation, Kubernetes delegates pod interface creation and configuration to JCNR-CNI. JCNR-CNI interacts with JCNR 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.

JCNR 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 JCNR 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.

2

CHAPTER

Install Cloud-Native Router on Baremetal Server

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 20 section to ensure the cluster has all the required configuration.
2. Download the tarball, **Juniper_Cloud_Native_Router_*release-number*.tgz**, to the directory of your choice. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

```
tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
```

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

```
cd Juniper_Cloud_Native_Router_release-number
```

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

5. View the contents in the current directory.

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

6. The JCNR container images are required for deployment. You may choose one of the following options:

   a. Download and deploy images from the Juniper repository—`enterprise-hub.juniper.net`. Review the "Configure Repository Credentials" on page 301 topic for instructions on how to configure repository credentials in the deployment helm chart.

   b. You can upload the JCNR images either to a local docker or to your own docker respository using the `docker load` command. The images are available in the **Juniper_Cloud_Native_Router_*release-number*/images** directory.

```
docker load -i images/jcnr-images.tar.gz
```

7. Enter the root password for your host server and your Juniper Cloud-Native Router license file into the **secrets/jcnr-secrets.yaml** file. You must enter the password and license in base64 encoded format.

   You can view the sample contents of the **jcnr-secrets.yaml** file below:

```
---
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>
```

To encode the password, create a file with the plain text password on a single line. Then issue the command:

```
base64 -w 0 rootPasswordFile
```

To encode the license, copy the license key into a file on your host server and issue the command:

```
base64 -w 0 licenseFile
```

You must copy the base64 outputs and paste them into the **secrets/jcnr-secrets.yaml** file in the appropriate locations.

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

Apply the **secrets/jcnr-secrets.yaml** to the Kubernetes system.

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

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

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

   See, "Customize JCNR Helm Chart for Baremetal Servers" on page 29 for descriptions of the helm chart configurations.

9. Optionally, customize JCNR configuration.

   See, "Customize JCNR Configuration " on page 54 for creating and applying the cRPD customizations.

10. Label the nodes to which JCNR mut be installed based on the `nodeaffinity`, if defined in the `values.yaml`. For example:

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

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

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

```
helm install jcnr .
```

> **NOTE**: The telemetry exporter in the cRPD pod is disabled by default. Specify the `--set jcnr-cni.telemetryExporter.enable=true` parameter in the `helm install` command to enable the cRPD telemetry exporter deployment. To learn more about cRPD telemetry, see *Telemetry Capabilities of Cloud-Native Router*.

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

12. 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-23.3.0               23.3.0
```

Verify Installation

This section enables you to confirm a successful JCNR deployment.

1. Verify the state of the JCNR pods by issuing the `kubectl get pods -A` command.

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

```
kubectl get pods -A
```

```
NAMESPACE         NAME                                    READY   STATUS
RESTARTS        AGE
contrail-deploy   contrail-k8s-deployer-579cd5bc74-g27gs   1/1     Running
0               103s
contrail          contrail-vrouter-masters-lqjqk          3/3     Running
0               87s
jcnr              kube-crpd-worker-sts-0                  1/1     Running
0               103s
jcnr              syslog-ng-ds5qd                         1/1     Running
0               103s
kube-system       calico-kube-controllers-5f4fd8666-m78hk 1/1     Running
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 JCNR daemonsets by issuing the `kubectl get ds -A` command.

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

```
kubectl get ds -A
```

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

3. Verify the JCNR statefulsets by issuing the `kubectl get statefulsets -A` command.

   The command output provides the statefulsets.

```
kubectl get statefulsets -A
```

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

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

   a. View the *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 JCNR 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 JCNR 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 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 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
            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

This section lists the host system requirements for installing the cloud-native router on a baremetal server.

Table 1: Cloud-Native Router Minimum Host System Requirements

Component	Value/Version	Notes
CPU	Intel x86	The tested CPU is Intel Xeon Gold 6212U 24-core @2.4 GHz

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

Component	Value/Version	Notes
Host OS	RedHat Enterprise Linux	Version 8.4, 8.5, 8.6
	Rocky Linux	8.6
Kernel Version	RedHat Enterprise Linux (RHEL): 4.18.X Rocky Linux: 4.18.X	The tested kernel version for RHEL is 4.18.0-305.rt7.72.el8.x86_64 The tested kernel version for Rocky Linux is 4.18.0-372.19.1.rt7.176.el8_6.x86_64 and 4.18.0-372.32.1.rt7.189.el8_6.x86_64
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	
IAVF driver	Version 4.8.2	
ICE_COMMS	Version 1.3.35.0	

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

Component	Value/Version	Notes
ICE	Version 1.11.20.13	ICE driver is used only with the Intel E810 NIC
i40e	Version 2.22.18.1	i40e driver is used only with the Intel XL710 NIC
Kubernetes (K8s)	Version 1.22.x, 1.23.x, 1.25x	The tested K8s version is 1.22.4. K8s version 1.22.2 also works. JCNR 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	Version 3.22.x	
Multus	Version 3.8	
Helm	3.9.x	
Container-RT	Docker CE 20.10.11, crio 1.25x	

Resource Requirements

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

Table 2: Cloud-Native Router Resource Requirements

Resource	Value	Usage Notes
Data plane forwarding cores	2 cores (2P + 2S)	
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	Add `GRUB_CMDLINE_LINUX_DEFAULT` values in **/etc/default/grub** on the host. For example: `GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
		Update grub and reboot the host. For example:
		`grub2-mkconfig -o /boot/grub2/grub.cfg`
		Verify the hugepage is set by executing the following commands:
		`cat /proc/cmdline`
		`grep -i hugepages /proc/meminfo`

JCNR Controller cores	.5	
JCNR vRouter Agent cores	.5	

Miscellaneous Requirements

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

Table 3: Miscellaneous Requirements

Cloud-Native Router Release Miscellaneous Requirements

Enable VLAN driver at system boot using the command:
`cat /etc/modules-load.d/vlan.conf`
`8021q`
Verify by executing the command:
`lsmod | grep 8021q`

Enable VFIO-PCI driver at system boot.
Enable the host with SR-IOV and VT-d in the system's BIOS.

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

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

Table 3: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

Additional kernel modules need to be loaded on the host before deploying JCNR in L3 mode. These modules are usually available in `linux-modules-extra` or `kernel-modules-extra` packages. Run the following commands to add the kernel modules:
 	
`cat /etc/modules-load.d/crpd.conf`
`tun`
`fou`
`fou6`
`ipip`
`ip_tunnel`
`ip6_tunnel`
`mpls_gso`
`mpls_router`
`mpls_iptunnel`
`vrf`
`vxlan`
 	
NOTE: Applicable for L3 deployments only.

Run the `ip fou add port 6635 ipproto 137` command on the Linux host to enable kernel based forwarding.

Table 3: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

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

Table 3: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

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

Port Requirements

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

Table 4: Cloud-Native Router Listening Ports

Protocol	Port	Description
TCP	8085	vRouter introspect–Used to gain internal statistical information about vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from JCNR control plane
TCP	9091	vRouter health check–cloud-native router checks to ensure **contrail-vrouter-dpdk** process is running, etc.
TCP	50052	gRPC port–JCNR listens on both IPv4 and IPv6
TCP	8081	JCNR Deployer Port

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

Protocol	Port	Description
TCP	22	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	50051	**jsd** on cRPD
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

To deploy JCNR on BMS you can download the helm charts from the Juniper Support Site.

JCNR Licensing

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

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

Customize JCNR Helm Chart for Baremetal Servers

IN THIS SECTION

Read this topic to learn about the deployment configuration available for the Juniper Cloud-Native Router on Baremetal Servers.

You can deploy and operate Juniper Cloud-Native Router in the L2, L3, or L2-L3 mode on a baremetal 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 Attributes and Descriptions

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.

Table 5: Helm Chart Attributes and Descriptions

Key	Additional Key Configuration	Description
registry		Defines the docker registry for the vRouter, cRPD and jcnr-cni 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 vRouter, cRPD and jcnr-cni container images. This is a global key and takes precedence over "repository" paths under "common" section. The default value is `jcnr-container-prod/`.
imagePullSecret		(Optional) Defines the registry authentication credentials. You can configure credentials to either the Juniper repository or your private registry.
	registryCredentials	Base64 representation of your Docker registry credentials. View the "Configure Repository Credentials" on page 301 topic for more information.
	secretName	Name of the secret object that will be created.
common		Defines repsitory paths and tags for the vRouter, cRPD and jcnr-cni container images. Use default unless using a private registry.
	repository	Defines the repository path. The default value is `atom-docker/cn2/bazel-build/dev/`. The global repository key takes precedence if defined.
	tag	Defines the image tag. The default value is configured to the appropriate tag number for the JCNR release version.

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

Key	Additional Key Configuration	Description
replicas		(Optional) Indicates the number of replicas for cRPD. If the value is not specified, then the default value 1 is considered. The value for this key must be specified for multi-node clusters. The value must be equal to the number of nodes to which JCNR must be deployed.
storageClass		Not applicable for non-cloud deployments.
awsregion		Not applicable for non-EKS deployments.
noLocalSwitching		(Optional) Prevents interfaces in a bridge domain from transmitting and receiving ethernet frame copies. Enter one or more comma separated VLAN IDs to ensure that the interfaces belonging to the VLAN IDs do not transmit frames to one another. This key is specific for L2 and L2-L3 deployments. Enabling this key provides the functionality on all access interfaces. For enabling the functionality on trunk interfaces, configure the no-local-switching key in the fabricInterface key.

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

Key	Additional Key Configuration	Description

fabricInterface		Provide a list of interfaces to be bound to the DPDK. You can also provide subnets instead of interface names. If both the interface name and the subnet are specified, then the interface name takes precedence over subnet/gateway combination. The subnet/gateway combination is useful when the interface names vary in a multi-node cluster.
		
		NOTE:
		• When all the interfaces have an `interface_mode` key configured, then the mode of deployment would be L2.
		• When one or more interfaces have an `interface_mode` key configured along with the rest of the interfaces not having the `interface_mode` key, then the mode of deployment would be L2-L3.
		• When none of the interfaces have the `interface_mode` key configured, then the mode of deployment would be L3.
		
		For example:
		
		`# L2 only`
		`- eth1:`
		`ddp: "auto"`
		`interface_mode: trunk`
		`vlan-id-list: [100, 200, 300, 700-705]`
		`storm-control-profile: rate_limit_pf1`
		`native-vlan-id: 100`
		`no-local-switching: true`
		
		`# L3 only`
		`- eth1:`
		`ddp: "off"`
		
		`# L2L3`
		`- eth1:`
		`ddp: "auto"`
		`- eth2:`
		`ddp: "auto"`
		`interface_mode: trunk`

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

Key	Additional Key Configuration	Description

		`vlan-id-list: [100, 200, 300, 700-705]`
		`storm-control-profile: rate_limit_pf1`
		`native-vlan-id: 100`
		`no-local-switching: true`

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

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

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

Key	Additional Key Configuration	Description
	storm-control-profile	Use `storm-control-profile` to associate appropriate storm control profile for the interface. Profiles are defined under `jcnr-vrouter.stormControlProfiles`.

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

	no-local-switching	Prevents interfaces from communicating directly with each other if the no-local-switching statement is configured. Allowed values are true or false.
fabricWorkloadInterface		(Optional) Defines the interfaces to which different workloads are connected. They can be software-based or hardware-based interfaces.
log_level		Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave the log_level set to the default INFO unless instructed to change it by Juniper support.
log_path		The defined directory stores various JCNR related descriptive logs such as contrail-vrouter-agent.log, contrail-vrouter-dpdk.log, etc. The default value is /var/log/jcnr/.
syslog_notifications		Indicates the absolute path to the file that stores syslog-ng generated notifications in JSON format. The default value is /var/log/jcnr/jcnr_notifications.json.

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

Key	Additional Key Configuration	Description
corePattern		Indicates the core pattern to denote how the core file is generated. If this configuration is left blank, then JCNR pods will not overwrite the default pattern. **NOTE**: Set the corePattern value on host before deploying JCNR. You may change the value in `/etc/sysctl.conf`. For example, `kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`
coreFilePath		Indicates the path for the core file. If the value is left blank, then vRouter considers /var/crashes as the default value.

nodeAffinity		(Optional) Defines labels on nodes to determine where to place the vRouter pods.
		
		By default the vRouter pods are deployed to all worker 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 JCNR must 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 cni path with the path in your distribution e.g. /var/opt/cni/bin.

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

Key	Additional Key Configuration	Description
grpcTelemetryPort		(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50051.
grpcVrouterPort		(Optional) Enter a value for this parameter to override vRouter gRPC server default port of 50052.
vRouterDeployerPort		(Optional) Default value is 8081. Configure to override if the default port is unavailable.
restoreInterfaces		Set the value of this key to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts.
bondInterfaceConfigs		(Optional) Enable bond interface configurations only for L2 or L2-L3 deployments.
	name	Name of the bond interface.
	mode	Default value is 1 (Active_Backup)
	slaveInterfaces	Fabric interfaces to be aggregated.
	primaryInterface	(Optional) Define primary interface for a bond. If this key is not configured, then the primary interface option is disabled.
mtu		Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default value is 9000.
cpu_core_mask		Indicates the vRouter forward core mask. If qos is enabled, you will need to allocate 4 CPU cores (primary and siblings).
stormControlProfiles		Configure the rate limit profiles for BUM traffic on fabric interfaces in bytes per second.

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

Key	Additional Key Configuration	Description
dpdkCommandAdditionalArgs		Pass any additional dpdk cmd line parameters. The --yield_option 0 is set by default and it implies the dpdk forwarding cores will not yield the cpu cores it is assigned to. Additional 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"`
ddp		(Optional) Indicates the global Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at NIC for traffic like GTPU, SCTP, etc. For a bond interface, all slave interface NICs must support DDP for the DDP configuration to be enabled. Setting options include auto, on, or off. The default setting is off. **NOTE**: The interface level `ddp` takes precedence over the global `ddp` configuration.
qosEnable		Set to true or false to enable or disable QoS. **NOTE**: QoS is not supported on Intel X710 NIC.
vrouter_dpdk_uio_driver		The uio driver is `vfio-pci`.
agentModeType		Can be dpdk or xdp. Setting agentModeType to dpdk will bringup dpdk datapath. Setting agentModeType to xdp uses ebpf. The default value is 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.
persistConfig		Set this flag to true if you wish jcnr-cni generated pod configuration to persist even after uninstallation. The option must be set only for L2 mode. The default value is false.

Helm Chart for L2 Only Deployment

A working L2 only helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred

  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

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

```
  # storageClass: Name of the storage class for cRPD. This option is must for
  # cloud deployments such as AWS where gp2 can be used
  #storageClass: gp2
  # Set AWS Region for AWS deployments
  #awsregion: us-east-1
  #noLocalSwitching: [700]

  # fabricInterface: provide a list of interfaces to be bound to dpdk
  # You can also provide subnets instead of interface names. Interfaces name take precedence over
  # Subnet/Gateway combination if both specified (although there is no reason to specify both)
  # Subnet/Gateway combination comes handy when the interface names vary in a multi-node cluster
  fabricInterface:
  ########################
  # L2 only
  - bond0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
  - ens2f2v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
  - ens2f3v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
  - ens1f0v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
      ddp: "auto"
      interface_mode: trunk
      storm-control-profile: rate_limit_pf1
      native-vlan-id: 1110
      no-local-switching: true

  ########################
  # L3  only
  #- eth11:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  ########################
```

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

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
  #- subnet: 10.40.1.0/24
  #  gateway: 10.40.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off
  #- subnet: 192.168.1.0/24
  #  gateway: 192.168.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off

  ###################################
  # fabricWorkloadInterface is applicable only for Pure L2 deployments
  #
  fabricWorkloadInterface:
  - ens1f1v0:
      interface_mode: access
      vlan-id-list: [1110]
 ##########################

  # 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: ""

  # path for the core file; vrouter considers /var/crashes as default value if not specified
  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
50051
  #grpcTelemetryPort: 50055

  # 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:
```

```
  # 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:
      - "ens2f0v0"
      - "ens2f1v0"
  #    primaryInterface: "ens2f0v0"
  # MTU for all physical interfaces( all VF's and  PF's)
  mtu: "9000"

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Helm Chart for L3 Only Deployment

A working L3 only helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred

   common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
```

```
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

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

  # storageClass: Name of the storage class for cRPD. This option is must for
  # cloud deployments such as AWS where gp2 can be used
  #storageClass: gp2
  # Set AWS Region for AWS deployments
  #awsregion: us-east-1
  #noLocalSwitching: [700]

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

```
  #- bond0:
  #    ddp: "auto" # auto/on/off  # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    #native-vlan-id: 100
  #    #no-local-switching: true

  ########################
  # L3  only
  - ens2f2:
      ddp: "auto"
  - ens1f1:
      ddp: "auto"
  ########################

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

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
  #- subnet: 10.40.1.0/24
  #  gateway: 10.40.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off
  #- subnet: 192.168.1.0/24
  #  gateway: 192.168.1.1
```

```
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off

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

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

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

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

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

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

```
  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  #cni_bin_dir:  /var/lib/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
  #vRouterDeployerPort: 8082


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

 # Enable bond interface configurations L2 only or L2 L3 deployment

  #bondInterfaceConfigs:
  #  - name: "bond0"
  #    mode: 1             # ACTIVE_BACKUP MODE
  #    slaveInterfaces:
  #    - "enp59s0f0v0"
  #    - "enp59s0f0v1"
  #    primaryInterface: "enp59s0f0v0"
  # MTU for all physical interfaces( all VF's and  PF's)
  mtu: "9000"

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

  # rate limit profiles for bum traffic on fabric interfaces in bytes per second
  stormControlProfiles:
    rate_limit_pf1:
```

```
      bandwidth:
        level: 0
    #rate_limit_pf2:
    #  bandwidth:
    #    level: 0

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Helm Chart for L2-L3 Deployment

A working L2-L3 helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred

  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

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

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

```
  # Set AWS Region for AWS deployments
  #awsregion: us-east-1
  #noLocalSwitching: [700]

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

  ##########################
  # L3  only
  #- eth11:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
```

```
off; default: off
  ########################

  # L2L3
    - bond0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
      storm-control-profile: rate_limit_pf1
      ddp: "auto"
  - ens2f0v1:
      ddp: "auto"
  - enp179s0f1v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
      ddp: "auto"
  - enp179s0f1v1:
      ddp: "auto"
  ##################################

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
  #- subnet: 10.40.1.0/24
  #  gateway: 10.40.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off
  #- subnet: 192.168.1.0/24
  #  gateway: 192.168.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off

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

```
  #    vlan-id-list: [800, 900]
 ##########################

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

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

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

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

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

  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  #cni_bin_dir:  /var/lib/cni/bin

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

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

```
  #grpcVrouterPort: 50060

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
  #vRouterDeployerPort: 8082


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

 # Enable bond interface configurations L2 only or L2 L3 deployment

  bondInterfaceConfigs:
     - name: "bond0"
       mode: 1             # ACTIVE_BACKUP MODE
       slaveInterfaces:
       - "ens2f0v0"
       - "ens2f1v0"
  #    primaryInterface: "enp59s0f0v0"
  # MTU for all physical interfaces( all VF's and  PF's)
  mtu: "9000"

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Customize JCNR Configuration

SUMMARY

Read this topic to understand how to customize JCNR configuration using a ConfigMap.

IN THIS SECTION

JCNR ConfigMap

Starting with Juniper Cloud-Native Router (JCNR) Release 23.3, JCNR supports customizing configuration using a ConfigMap when deployed in L3 mode. In cloud-based deployments, in the event of a node failure, the JCNR pods may be spawned on newer or different nodes. A ConfigMap decouples the configuration parameters from node names and is based on node labels instead. This enables the JCNR CNI deployer to consume the configuration parameters, apply them to the cRPD configuration template and render the configuration, as long as a matching label is available for the node.

A ConfigMap is an API object to store data in key-values pairs. A ConfigMap defines per node variables that are consumed by nodes matching the label. The key-value pairs are used to render the configuration via a go template. The configured template must be available in the `Juniper_Cloud_Native_Router_`*`release_number`*`/helmchart/charts/jcnr-cni/files/` directory for the configuration to be applied to the cRPD pods.

NOTE: You must apply the ConfigMap before installing JCNR to create cRPD pods with custom configuration. The cRPD pod must be deleted and respawned should you wish to apply the configuration parameters any time after JCNR installation. The configuration parameters are applied by default to any newly spawned cRPD pods. The JCNR customization via ConfigMap is optional.

NOTE: JCNR also supports customization via node annotations for backward compatibility with previous releases. Considering that node annotations are coupled with node names, it is highly recommended to customize JCNR via ConfigMaps, specifically for cloud deployments. Refer to Customize JCNR Configuration using node annotations for more information.

Configuration Example

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

You define the key-value pair for different node labels in your cluster. An example of the `jcnr-params-configmap.yaml` file is provided below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: jcnr-params
```

```
  namespace: jcnr
data:
  jcnr1: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0001.00",
      "IPv4LoopbackAddr": "110.1.1.2",
      "srIPv4NodeIndex": "2000",
      "srIPv6NodeIndex": "3000",
      "BGPIPv4Neighbor": "110.1.1.254",
      "BGPLocalAsn": "64512"
    }
  jcnr2: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0000.00",
      "IPv4LoopbackAddr": "110.1.1.3",
      "srIPv4NodeIndex": "2001",
      "srIPv6NodeIndex": "3001",
      "BGPIPv4Neighbor": "110.1.2.254",
      "BGPLocalAsn": "64512"
    }
```

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

```
apply-groups [custom];
groups {
    custom {
        interfaces {
            lo0 {
                unit 0 {
                {{if .Params.isoLoopbackAddr}}
                    family iso {
                        address {{.Params.isoLoopbackAddr}};
                    }
                {{end}}
                    family inet {
                        address {{.Params.IPv4LoopbackAddr}};
                    }
                }
            }
        }
        routing-options {
```

```
            router-id {{.Params.IPv4LoopbackAddr}}
            route-distinguisher-id {{.Params.IPv4LoopbackAddr}}
        }
        protocols {
            isis {
                interface all;
                {{if and .Env.SRGB_START_LABEL .Env.SRGB_INDEX_RANGE}}
                source-packet-routing {
                    srgb start-label {{.Env.SRGB_START_LABEL}} index-range
{{.Env.SRGB_INDEX_RANGE}};
                    node-segment {
                        {{if .Params.srIPv4NodeIndex}}
                        ipv4-index {{.Params.srIPv4NodeIndex}};
                        {{end}}
                        {{if .Params.srIPv6NodeIndex}}
                        ipv6-index {{.Params.srIPv6NodeIndex}};
                        {{end}}
                    }
                }
                {{end}}
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
        policy-options {
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then community add udp;
            }
            community udp members encapsulation:0L:13;
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address {{.Params.IPv4LoopbackAddr}};
                    local-as {{.Params.BGPLocalAsn}};
                    neighbor {{.Params.BGPIPv4Neighbor}};
```

```
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                }
            }
        }
        routing-options {
                dynamic-tunnels {
                dyn-tunnels {
                    source-address {{.Params.IPv4LoopbackAddr}};
                    udp;
                    destination-networks {{.Params.BGPIPv4Neighbor}}/32;
                }
            }
        }
    }
}
```

NOTE: You can define additional cRPD configuration hierarchies in the template. The values to be rendered from the ConfigMap defined in the `jcnr-params-configmap.yaml` must be defined as {{.Params.*var-name*}}. Any environment variables, such as variables defined in `values.yaml`, must be defined as {{.Env.*variable_name*}}.

Complete the following steps to apply the customizations.

1. Label each node based on the keys used in the ConfigMap.

```
kubectl label nodes <node_name1> jcnr.juniper.net/params-profile=jcnr1
kubectl label nodes <node_name2> jcnr.juniper.net/params-profile=jcnr2
```

2. Apply the ConfigMap to the cluster nodes using the command provided below:

```
# kubectl apply -f jcnr-params-configmap.yaml
configmap/jcnr-params created
```

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

```
# cp Juniper_Cloud_Native_Router_release_number/helmchart/cRPD_examples/jcnr-cni-custom-
config-cm.tmpl Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/
#
```

4. Deploy the cloud-native router components, including the cRPD. Once the installation completes, *access the cRPD CLI* and issue the `show configuration | display set` command in the `cli` mode to view the custom configuration you applied.

```
root@jcnr-01> show configuration
## Last commit: 2023-06-23 08:30:42 EDT by root
version 20230608.143922_builder.r1342735;
groups {
    base { /* OMITTED */ };
    custom {
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 110.1.1.2/32;
                    }
                    family iso {
                        address 49.0004.1000.0000.0001.00;
                    }
                }
            }
        }
        policy-options {
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then {
                    community add udp;
                }
            }
            community udp members encapsulation:0L:13;
        }
        routing-options {
            route-distinguisher-id 110.1.1.2;
            router-id 110.1.1.2;
```

```
            dynamic-tunnels {
                dyn-tunnels {
                    source-address 110.1.1.2;
                    udp;
                    destination-networks {
                        110.1.1.254/32;
                    }
                }
            }
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address 110.1.1.2;
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                    local-as 64512;
                    neighbor 110.1.1.254;
                }
            }
            isis {
                interface all;
                source-packet-routing {
                    srgb start-label 400000 index-range 4000;
                    node-segment {
                        ipv4-index 2000;
                        ipv6-index 3000;
                    }
                }
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
```

```
    }
    cni { /* OMITTED */ };
    internal { /* OMITTED */ };
}
apply-groups [ custom base internal ];
```

Modifying the ConfigMap

If you wish to change the ConfigMap any time after JCNR installation, you must delete the cRPD pod and respawn it using the following command:

```
kubectl patch sts kube-crpd-worker-sts -n jcnr -p '{"spec":{"template":{"metadata":
{"annotations":{"configmap-reload/timestamp":"'$(date +%s)'"}}}}}'
```

This triggers a rolling restart of all cRPD pods. Alternatively, you can identify the cRPD pods on nodes for which the ConfigMap has changed and manually delete the pod. The ConfigMap changes will be applied automatically to any respawned pods.

Troubleshooting

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

CHAPTER 3

Install Cloud-Native Router on Red Hat OpenShift

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 73 to ensure the cluster has all the required configuration.
2. Download the tarball, **Juniper_Cloud_Native_Router_*release-number*.tgz**, to the directory of your choice. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

```
tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
```

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

```
cd Juniper_Cloud_Native_Router_release-number
```

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

5. View the contents in the current directory.

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

6. The JCNR container images are required for deployment. You may choose one of the following options:

   a. Download and deploy images from the Juniper repository—`enterprise-hub.juniper.net`. Review the "Configure Repository Credentials" on page 301 topic for instructions on how to configure repository credentials in the deployment helm chart.

   b. You can upload the JCNR images to a local registry. The images are available in the **Juniper_Cloud_Native_Router_*release-number*/images** directory.

7. Enter the root password for your host server and your Juniper Cloud-Native Router license file into the **secrets/jcnr-secrets.yaml** file. You must enter the password and license in base64 encoded format.

   You can view the sample contents of the **jcnr-secrets.yaml** file below:

```
---
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>
```

To encode the password, create a file with the plain text password on a single line. Then issue the command:

```
base64 -w 0 rootPasswordFile
```

To encode the license, copy the license key into a file on your host server and issue the command:

```
base64 -w 0 licenseFile
```

You must copy the base64 outputs and paste them into the **secrets/jcnr-secrets.yaml** file in the appropriate locations.

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

Apply the **secrets/jcnr-secrets.yaml** to the Kubernetes system.

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

> **NOTE**: Starting with JCNR Release 23.2, the JCNR license format has changed. Request a new license key from the JAL portal before deploying or upgrading to 23.2 or newer releases.

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

   See, "Customize JCNR Helm Chart for OpenShift Deployment" on page 84 for descriptions of the helm chart configurations.

9. Optionally, customize JCNR configuration.

   See, "Customize JCNR Configuration " on page 54 for creating and applying the cRPD customizations.

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

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

```
helm install jcnr .
```

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

11. 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-23.3.0                23.3.0
```

Verify Installation

This section enables you to confirm a successful JCNR deployment.

1. Verify the state of the JCNR 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                                               contrail-vrouter-nodes-
dt8sx                                    3/3     Running            0              16d
jcnr                                                   kube-crpd-worker-
sts-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 JCNR daemonsets by issuing the `kubectl get ds -A` command.

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

```
kubectl get ds -A
```

```
NAMESPACE                              NAME                          DESIRED   CURRENT
READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR                                                  AGE
contrail                               contrail-vrouter-masters      0         0
0       0            0
<none>                                                    16d
contrail                               contrail-vrouter-nodes        2         2
2       2            2
<none>                                                    16d
jcnr                                   syslog-ng                     2         2
2       2            2
<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 JCNR statefulsets by issuing the `kubectl get statefulsets -A` command.

   The command output provides the statefulsets.

```
kubectl get statefulsets -A
```

```
NAMESPACE             NAME                  READY   AGE
jcnr                  kube-crpd-worker-sts  2/2     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 JCNR 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 JCNR 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

JCNR Licensing | 83

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

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

Table 6: Cloud-Native Router Minimum Host System Requirements

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 4.12	
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 4.000x800144111.3236.0 • Intel XL710 with Firmware 9.00 0x8000cead 1.3179.0	
IAVF driver	Version 4.5.3.1	
ICE_COMMS	Version 1.3.35.0	

Table 6: Cloud-Native Router Minimum Host System Requirements *(Continued)*

Component	Value/Version	Notes
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	4.12	The tested versions are: Client Version: 4.12.0-202301042257.p0.g854f807.assembly.stream-854f807 Kustomize Version: v4.5.7 Server Version: 4.12.0 Kubernetes Version: v1.25.4+77bec7a
OVN-Kubernetes CNI		
Multus	Version 3.8	
Helm	3.12.x	
Container-RT	crio 1.25x	

Resource Requirements

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

Table 7: Cloud-Native Router Resource Requirements

Resource	Value	Usage Notes
Data plane forwarding cores	2 cores (2P + 2S)	
Service/Control Cores	0	

Table 7: Cloud-Native Router Resource Requirements *(Continued)*

Resource	Value	Usage Notes

UIO Driver	VFIO-PCI	To enable, follow the steps below:
		Create a Butane config file, `100-worker-vfiopci.bu`, binding the PCI device to the VFIO driver.
		
		`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`
		
		Create and apply the machine config:
		
		`$ butane 100-worker-vfiopci.bu -o 100-worker-vfiopci.yaml`
		
		`$ oc apply -f 100-worker-vfiopci.yaml`

Table 7: Cloud-Native Router Resource Requirements *(Continued)*

Resource	Value	Usage Notes

Hugepages (1G)	6 Gi	Configure hupages 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=8`
		`  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 7: Cloud-Native Router Resource Requirements *(Continued)*

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

Miscellaneous Requirements

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

Table 8: Miscellaneous Requirements

Cloud-Native Router Release Miscellaneous Requirements

Enable VLAN driver at system boot using the command:
`cat /etc/modules-load.d/vlan.conf`
`8021q`
Verify by executing the command:
`lsmod | grep 8021q`

Enable VFIO-PCI driver at system boot
Enable the host with SR-IOV and VT-d in the system's BIOS.

Table 8: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

Set IOMMU and IOMMU-PT
Create a MachineConfig object that defined a kernel argument:
 	
`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. For example: `ip link set <interfacename> vf 1 spoofcheck off`. **NOTE**: Applicable only on L2 deployments.
Set trust on VFs allocated to JCNR. For example: `ip link set <interfacename> vf 1 trust on` **NOTE**: Applicable only on L2 deployments.

Table 8: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

Additional kernel modules need to be loaded on the host before deploying JCNR in L3 mode. These modules are usually available in `linux-modules-extra` or `kernel-modules-extra` packages. Run the following commands to add the kernel modules:
 	
`cat /etc/modules-load.d/crpd.conf`
`tun`
`fou`
`fou6`
`ipip`
`ip_tunnel`
`ip6_tunnel`
`mpls_gso`
`mpls_router`
`mpls_iptunnel`
`vrf`
`vxlan`
 	
NOTE: Applicable for L3 deployments only.

Run the `ip fou add port 6635 ipproto 137` command on the Linux host to enable kernel based forwarding.

Table 8: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

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

Table 8: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

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

Port Requirements

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

Table 9: Cloud-Native Router Listening Ports

Protocol	Port	Description
TCP	8085	vRouter introspect–Used to gain internal statistical information about vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from JCNR control plane
TCP	9091	vRouter health check–cloud-native router checks to ensure **contrail-vrouter-dpdk** process is running, etc.
TCP	50052	gRPC port–JCNR listens on both IPv4 and IPv6
TCP	8081	JCNR Deployer Port

Table 9: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
TCP	22	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	50051	**jsd** on cRPD
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

To deploy JCNR on OCP you can download the helm charts from the Juniper Support Site.

JCNR Licensing

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

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

Customize JCNR 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.

Table 10: Helm Chart Attributes and Descriptions

Key	Additional Key Configuration	Description
registry		Defines the docker registry where the vRouter, cRPD and jcnr-cni container images are hosted. The default value is `enterprise-hub.juniper.net`.
repository		(Optional) Defines the repository path for the vRouter, cRPD and jcnr-cni container images. This is a global key and takes precedence over "repository" paths under "common" section.
imagePullSecret		(Optional) Defines the registry authentication credentials. You can configure credentials to either the Juniper repository or your private registry.
	registryCredentials	Base64 representation of your Docker registry credentials. View the "Configure Repository Credentials" on page 301 topic for more information.
	secretName	Name of the secret object that will be created.
common		Defines repsitory paths and tags for the vRouter, cRPD and jcnr-cni container images. Use default unless using a private registry.
	repository	Defines the repository path. The default value is `atom-docker/cn2/bazel-build/dev/`. The global repository key takes precedence if defined.
	tag	Defines the image tag. The default value is configured to the appropriate tag number for the JCNR release version.
replicas		(Optional) Indicates the number of replicas for cRPD. If the value is not specified, then the default value 1 is considered. The value for this key must be specified for multi-node clusters.
storageClass		Not applicable for OCP deployment.

Table 10: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
awsregion		Not applicable for OCP deployment.
noLocalSwitching		(Optional) Prevents interfaces in a bridge domain from transmitting and receiving ethernet frame copies. Enter one or more comma separated VLAN IDs to ensure that the interfaces belonging to the VLAN IDs do not transmit frames to one another. This key is specific for L2 and L2-L3 deployments. Enabling this key provides the functionality on all access interfaces. For enabling the functionality on trunk interfaces, configure the no-local-switching key in the fabricInterface key.

Table 10: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description

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

Table 10: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
		- eth2: ddp: "off" # L2L3 - eth1: ddp: "auto" - eth2: ddp: "auto" interface_mode: trunk vlan-id-list: [100, 200, 300, 700-705] storm-control-profile: rate_limit_pf1 native-vlan-id: 100 no-local-switching: true
	subnet	An alternative mode of input for interface names. For example: - subnet: 10.40.1.0/24 gateway: 10.40.1.1 ddp: "off" The subnet option is applicable only for L3 interfaces. With the subnet mode of input, interfaces are auto-detected in each subnet. Specify either subnet/gateway or the interface name. Do not configure both. The subnet/gateway form of input is particularly helpful in environments where the interface names vary for a multi-node K8s cluster.
	ddp	(Optional) Indicates the interface-level Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at NIC for traffic like GTPU, SCTP, etc. For a bond interface, all slave interface NICs must support DDP for the DDP configuration to be enabled. Setting options include auto, on, or off. The default setting is off. **NOTE**: The subnet/interface level ddp takes precedence over the global ddp configuration.

Table 10: Helm Chart Attributes and Descriptions *(Continued)*

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

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

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

Table 10: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
log_path		The defined directory stores various JCNR related descriptive logs such as contrail-vrouter-agent.log, contrail-vrouter-dpdk.log, etc. The default value is /var/log/jcnr/.
syslog_notifications		Indicates the absolute path to the file that stores syslog-ng generated notifications in JSON format. The default value is /var/log/jcnr/jcnr_notifications.json.
corePattern		Indicates the core pattern to denote how the core file is generated. If this configuration is left blank, then JCNR pods will not overwrite the default pattern. **NOTE**: Set the corePattern value on host before deploying JCNR. You may change the value in `/etc/sysctl.conf`. For example, `kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`
coreFilePath		Indicates the path for the core file. If the value is left blank, then vRouter considers /var/crashes as the default value.

nodeAffinity		(Optional) Defines labels on nodes to determine where to place the vRouter and pods. 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.
		On an OCP setup node affinity must be configured to bring up JCNR only on worker nodes.

	key	Key-value pair that represents a node label that must be matched to apply the node affinity.

Table 10: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	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 cni path with a path of your choice e.g. /var/opt/cni/bin. For Red Hat OpenShift the default CNI path should be configured to /var/lib/cni/bin, which is the default path on any OCP deployment. Leaving the path variable (cni_bin_dir) empty, isn't a viable option in OCP.
grpcTelemetryPort		(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50051.
grpcVrouterPort		(Optional) Enter a value for this parameter to override vRouter gRPC server default port of 50052.
vRouterDeployerPort		(Optional) Default value is 8081. Configure to override if the default port is unavailable.
restoreInterfaces		Set the value of this key to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts.
bondInterfaceConfigs		(Optional) Enable bond interface configurations only for L2 or L2-L3 deployments.
	name	Name of the bond interface.
	mode	Default value is 1 (Active_Backup)
	slaveInterfaces	Fabric interfaces to be aggregated.
	primaryInterface	(Optional) Define primary interface for a bond. If this key is not configured, then the primary interface option is disabled.
mtu		Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs).

Table 10: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
cpu_core_mask		Indicates the vRouter forward core mask. If qos is enabled, you will need to allocate 4 CPU cores (primary and siblings).
stormControlProfiles		Configure the rate limit profiles for BUM traffic on fabric interfaces in bytes per second.
	rate_limit_pf1	
	bandwidth	
	level	
dpdkCommandAdditionalArgs		Pass any additional dpdk cmd line parameters. The --yield_option 0 is set by default and it implies the dpdk forwarding cores will not yield the cpu cores it is assigned to. Additional 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"`
ddp		(Optional) Indicates the global Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at NIC for traffic like GTPU, SCTP, etc. For a bond interface, all slave interface NICs must support DDP for the DDP configuration to be enabled. Setting options include auto, on, or off. The default setting is off. **NOTE**: The interface level `ddp` takes precedence over the global `ddp` configuration.
qosEnable		Set to true or false to enable or disable QoS. **NOTE**: QoS is not supported on Intel X710 NIC.

Table 10: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
vrouter_dpdk_uio_driver		The uio driver is `vfio-pci`.
agentModeType		Can be dpdk or xdp. Setting agentModeType to dpdk will bringup dpdk datapath. Setting agentModeType to xdp uses ebpf. The default value is 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.
persistConfig		Set this flag to true if you wish jcnr-cni generated pod configuration to persist even after uninstallation. The option must be set only for L2 mode. The default value is false.

Sample Helm Charts

Helm Chart for L2 Only Deployment on Red Hat OpenShift

A working L2 only helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
```

```
#imagePullSecret:
  #registryCredentials: <base64-encoded-credential>
  #secretName: regcred

common:
  vrouter:
    repository: atom-docker/cn2/bazel-build/dev/x86_64/
    tag: R23.4-85
  crpd:
    repository: junos-docker-local/warthog/amd64/
    tag: 23.4R1.8
  jcnrcni:
    repository: junos-docker-local/warthog/amd64/
    tag: 23.4-20231215-50817e3
  telemetryExporter:
    repository: atom-docker/cn2/bazel-build/dev/x86_64/
    tag: R23.4-85

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

# storageClass: Name of the storage class for cRPD. This option is must for
# cloud deployments such as AWS where gp2 can be used
#storageClass: gp2
# Set AWS Region for AWS deployments
#awsregion: us-east-1
#noLocalSwitching: [700]

# fabricInterface: provide a list of interfaces to be bound to dpdk
# You can also provide subnets instead of interface names. Interfaces name take precedence over
# Subnet/Gateway combination if both specified (although there is no reason to specify both)
# Subnet/Gateway combination comes handy when the interface names vary in a multi-node cluster
fabricInterface:
########################
# L2 only
- bond0:
    interface_mode: trunk
    vlan-id-list: [1110-1141]
- ens2f2v0:
    interface_mode: trunk
    vlan-id-list: [1110-1141]
- ens2f3v0:
```

```
      interface_mode: trunk
      vlan-id-list: [1110-1141]
  - ens1f0v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
      ddp: "auto"
      interface_mode: trunk
      storm-control-profile: rate_limit_pf1
      native-vlan-id: 1110
      no-local-switching: true

  ########################
  # L3  only
  #- eth11:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  ########################

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

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

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

  ###################################
  # fabricWorkloadInterface is applicable only for Pure L2 deployments
  #
  fabricWorkloadInterface:
  - ens1f1v0:
      interface_mode: access
      vlan-id-list: [1110]
 ##########################

  # 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: ""

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

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

```
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  cni_bin_dir:  /var/lib/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
  #vRouterDeployerPort: 8082



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

 # Enable bond interface configurations L2 only or L2 L3 deployment

  bondInterfaceConfigs:
    - name: "bond0"
      mode: 1             # ACTIVE_BACKUP MODE
      slaveInterfaces:
      - "ens2f0v0"
      - "ens2f1v0"
  #    primaryInterface: "enp59s0f0v0"

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

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

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

```
    #rate_limit_pf2:
    #  bandwidth:
    #    level: 0

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Helm Chart for L3 Only Deployment on Red Hat OpenShift

A working L3 only helm chart sample is shown below. The configured sections are highlighted in **bold**:

```
##################################################################
#                Common Configuration (global vars)              #
##################################################################
```

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred

  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

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

  # storageClass: Name of the storage class for cRPD. This option is must for
  # cloud deployments such as AWS where gp2 can be used
  #storageClass: gp2
  # Set AWS Region for AWS deployments
  #awsregion: us-east-1
  #noLocalSwitching: [700]

  # fabricInterface: provide a list of interfaces to be bound to dpdk
  # You can also provide subnets instead of interface names. Interfaces name take precedence over
  # Subnet/Gateway combination if both specified (although there is no reason to specify both)
  # Subnet/Gateway combination comes handy when the interface names vary in a multi-node cluster
  fabricInterface:
```

```
  ########################
  # L2 only
  #- eth1:
  #    ddp: "auto"               # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  #- eth2:
  #    ddp: "auto"               # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [700]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  #- bond0:
  #    ddp: "auto" # auto/on/off  # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    #native-vlan-id: 100
  #    #no-local-switching: true

  ########################
  # L3  only
  - ens2f2:
      ddp: "auto"
  - ens1f1:
      ddp: "auto"
  ########################

  # L2L3
  #- eth1:
  #    ddp: "auto"               # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "auto"               # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
```

```
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  ###################################

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
  #- subnet: 10.40.1.0/24
  #  gateway: 10.40.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off
  #- subnet: 192.168.1.0/24
  #  gateway: 192.168.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off

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

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

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

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

  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  cni_bin_dir:  /var/lib/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
  #vRouterDeployerPort: 8082


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

 # Enable bond interface configurations L2 only or L2 L3 deployment

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

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

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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
```

Helm Chart for L2-L3 Deployment on Red Hat OpenShift

A working L2-L3 helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcreds

  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

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

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

```
  # Set AWS Region for AWS deployments
  #awsregion: us-east-1
  #noLocalSwitching: [700]

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

  ##########################
  # L3  only
  #- eth11:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
```

```
off; default: off
  ########################

  # L2L3
    - bond0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
      storm-control-profile: rate_limit_pf1
      ddp: "auto"
  - ens2f0v1:
      ddp: "auto"
  - enp179s0f1v0:
      interface_mode: trunk
      vlan-id-list: [1110-1141]
      ddp: "auto"
  - enp179s0f1v1:
      ddp: "auto"
  ##################################

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
  #- subnet: 10.40.1.0/24
  #  gateway: 10.40.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off
  #- subnet: 192.168.1.0/24
  #  gateway: 192.168.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off

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

```
  #    vlan-id-list: [800, 900]
 ##########################

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

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

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

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

  # 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

  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  cni_bin_dir:  /var/lib/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
```

```
8081
  #vRouterDeployerPort: 8082



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

 # Enable bond interface configurations L2 only or L2 L3 deployment

  bondInterfaceConfigs:
     - name: "bond0"
       mode: 1             # ACTIVE_BACKUP MODE
       slaveInterfaces:
       - "ens2f0v0"
       - "ens2f1v0"
  #    primaryInterface: "enp59s0f0v0"

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

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

```
  qosEnable: false

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Customize JCNR Configuration

SUMMARY

Read this topic to understand how to customize JCNR configuration using a ConfigMap.

IN THIS SECTION

JCNR ConfigMap

Starting with Juniper Cloud-Native Router (JCNR) Release 23.3, JCNR supports customizing configuration using a ConfigMap when deployed in L3 mode. In cloud-based deployments, in the event of a node failure, the JCNR pods may be spawned on newer or different nodes. A ConfigMap decouples the configuration parameters from node names and is based on node labels instead. This enables the

JCNR CNI deployer to consume the configuration parameters, apply them to the cRPD configuration template and render the configuration, as long as a matching label is available for the node.

A ConfigMap is an API object to store data in key-values pairs. A ConfigMap defines per node variables that are consumed by nodes matching the label. The key-value pairs are used to render the configuration via a go template. The configured template must be available in the `Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/` directory for the configuration to be applied to the cRPD pods.

NOTE: You must apply the ConfigMap before installing JCNR to create cRPD pods with custom configuration. The cRPD pod must be deleted and respawned should you wish to apply the configuration parameters any time after JCNR installation. The configuration parameters are applied by default to any newly spawned cRPD pods. The JCNR customization via ConfigMap is optional.

NOTE: JCNR also supports customization via node annotations for backward compatibility with previous releases. Considering that node annotations are coupled with node names, it is highly recommended to customize JCNR via ConfigMaps, specifically for cloud deployments. Refer to Customize JCNR Configuration using node annotations for more information.

Configuration Example

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

You define the key-value pair for different node labels in your cluster. An example of the `jcnr-params-configmap.yaml` file is provided below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: jcnr-params
  namespace: jcnr
data:
  jcnr1: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0001.00",
      "IPv4LoopbackAddr": "110.1.1.2",
```

```
      "srIPv4NodeIndex": "2000",
      "srIPv6NodeIndex": "3000",
      "BGPIPv4Neighbor": "110.1.1.254",
      "BGPLocalAsn": "64512"
    }
  jcnr2: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0000.00",
      "IPv4LoopbackAddr": "110.1.1.3",
      "srIPv4NodeIndex": "2001",
      "srIPv6NodeIndex": "3001",
      "BGPIPv4Neighbor": "110.1.2.254",
      "BGPLocalAsn": "64512"
    }
```

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

```
apply-groups [custom];
groups {
    custom {
        interfaces {
            lo0 {
                unit 0 {
                {{if .Params.isoLoopbackAddr}}
                    family iso {
                        address {{.Params.isoLoopbackAddr}};
                    }
                {{end}}
                    family inet {
                        address {{.Params.IPv4LoopbackAddr}};
                    }
                }
            }
        }
        routing-options {
            router-id {{.Params.IPv4LoopbackAddr}}
            route-distinguisher-id {{.Params.IPv4LoopbackAddr}}
        }
        protocols {
            isis {
                interface all;
```

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

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

NOTE: You can define additional cRPD configuration hierarchies in the template. The values to be rendered from the ConfigMap defined in the `jcnr-params-configmap.yaml` must be defined as {{.Params.*var-name*}}. Any environment variables, such as variables defined in `values.yaml`, must be defined as {{.Env.*variable_name*}}.

Complete the following steps to apply the customizations.

1. Label each node based on the keys used in the ConfigMap.

```
kubectl label nodes <node_name1> jcnr.juniper.net/params-profile=jcnr1
kubectl label nodes <node_name2> jcnr.juniper.net/params-profile=jcnr2
```

2. Apply the ConfigMap to the cluster nodes using the command provided below:

```
# kubectl apply -f jcnr-params-configmap.yaml
configmap/jcnr-params created
```

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

```
# cp Juniper_Cloud_Native_Router_release_number/helmchart/cRPD_examples/jcnr-cni-custom-
config-cm.tmpl Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/
#
```

4. Deploy the cloud-native router components, including the cRPD. Once the installation completes, *access the cRPD CLI* and issue the `show configuration | display set` command in the `cli` mode to view the custom configuration you applied.

```
root@jcnr-01> show configuration
## Last commit: 2023-06-23 08:30:42 EDT by root
version 20230608.143922_builder.r1342735;
groups {
    base { /* OMITTED */ };
    custom {
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 110.1.1.2/32;
                    }
                    family iso {
                        address 49.0004.1000.0000.0001.00;
                    }
                }
            }
        }
        policy-options {
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then {
                    community add udp;
                }
            }
            community udp members encapsulation:0L:13;
        }
        routing-options {
            route-distinguisher-id 110.1.1.2;
            router-id 110.1.1.2;
            dynamic-tunnels {
                dyn-tunnels {
                    source-address 110.1.1.2;
                    udp;
                    destination-networks {
                        110.1.1.254/32;
                    }
                }
```

```
            }
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address 110.1.1.2;
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                    local-as 64512;
                    neighbor 110.1.1.254;
                }
            }
            isis {
                interface all;
                source-packet-routing {
                    srgb start-label 400000 index-range 4000;
                    node-segment {
                        ipv4-index 2000;
                        ipv6-index 3000;
                    }
                }
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
    }
    cni { /* OMITTED */ };
    internal { /* OMITTED */ };
}
apply-groups [ custom base internal ];
```

Modifying the ConfigMap

If you wish to change the ConfigMap any time after JCNR installation, you must delete the cRPD pod and respawn it using the following command:

```
kubectl patch sts kube-crpd-worker-sts -n jcnr -p '{"spec":{"template":{"metadata":
{"annotations":{"configmap-reload/timestamp":"'$(date +%s)'"}}}}}'
```

This triggers a rolling restart of all cRPD pods. Alternatively, you can identify the cRPD pods on nodes for which the ConfigMap has changed and manually delete the pod. The ConfigMap changes will be applied automatically to any respawned pods.

Troubleshooting

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

CHAPTER 4

Install Cloud-Native Router on Amazon EKS

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 130 to ensure the setup has all the required configuration.
2. Download the tarball, **Juniper_Cloud_Native_Router_*<release-number>*.tgz**, to the directory of your choice. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
3. Expand the file **Juniper_Cloud_Native_Router_*<release-number>*.tgz**.

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

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

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

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

5. View the contents in the current directory.

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

6. Enter the root password for your host server and your Juniper Cloud-Native Router license file into the **secrets/jcnr-secrets.yaml** file. You must enter the password and license in base64 encoded format.

   You can view the sample contents of the **jcnr-secrets.yaml** file below:

```
---
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>
```

   To encode the password, create a file with the plain text password on a single line. Then issue the command:

```
base64 -w 0 rootPasswordFile
```

   To encode the license, copy the license key into a file on your host server and issue the command:

```
base64 -w 0 licenseFile
```

You must copy the base64 outputs and paste them into the **secrets/jcnr-secrets.yaml** file in the appropriate locations.

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

Apply the **secrets/jcnr-secrets.yaml** to the Kubernetes system.

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

> **NOTE**: Starting with JCNR Release 23.2, the JCNR license format has changed. Request a new license key from the JAL portal before deploying or upgrading to 23.2 or newer releases.

7. Create the "JCNR ConfigMap" on page 134 if using the Virtual Router Redundancy Protocol (VRRP) for your JCNR 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
```

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

   See, "Customize JCNR Helm Chart for EKS Deployment" on page 137 for descriptions of the helm chart configurations and a sample helm chart for EKS deployment.

9. Optionally, customize JCNR configuration.

   See, "Customize JCNR Configuration " on page 54 for creating and applying the cRPD customizations.

10. 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
```

11. Install the Amazon Elastic Block Storage (EBS) Container Storage Interface (CSI) driver.

12. Label the nodes to which JCNR must be installed based on the `nodeAffinity` defined in the `values.yaml`. For example:

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

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

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

```
helm install jcnr .
```

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

14. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

    Sample output:

```
NAME        NAMESPACE    REVISION        UPDATED
STATUS          CHART                     APP VERSION
jcnr        default      1               2023-09-22 06:04:33.144611017 -0400 EDT
deployed        jcnr-23.3.0               23.3.0
```

Install Juniper Cloud-Native Router Using AWS Marketplace Subscription

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 130 to ensure the setup has all the required configuration.

2. Configure AWS credentials using the command: `aws configure`.
3. Authenticate to the Amazon ECR repo.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-
stdin 709825985650.dkr.ecr.us-east-1.amazonaws.com
```

```
aws ecr get-login-password --region us-east-1 | helm registry login --username AWS --
password-stdin 709825985650.dkr.ecr.us-east-1.amazonaws.com
```

4. Download the helm package from the ECR repo.

```
helm pull oci://709825985650.dkr.ecr.us-east-1.amazonaws.com/juniper-networks/jcnr --
version 23.3.0
```

5. Expand the file **jcnr-23.3.0.tgz**.

```
tar xzvf jcnr-23.3.0.tgz
```

6. Change directory to `jcnr`.

```
cd jcnr
```

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

7. View the contents in the current directory.

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

8. Create a `jcnr_secrets.yaml` file to define the root password for your host server and your Juniper Cloud-Native Router license. You must enter the password and license in base64 encoded format.

   You can view the sample contents of the **jcnr-secrets.yaml** file below:

```
---
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>
```

The manifest creates the `jcnr` namespace, a `jcnr-secrets` secret with the root password and cRPD license.

To encode the password, create a file with the plain text password on a single line. Then issue the command:

```
base64 -w 0 rootPasswordFile
```

To encode the license, copy the license key onto your host server and issue the command:

```
base64 -w 0 licenseFile
```

You must copy the base64 outputs and paste them into the **secrets/jcnr-secrets.yaml** file in the appropriate locations.

NOTE:
jcnr-secrets.yamljcnr-secrets.yaml

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

9. Apply the `jcnr-secrets.yaml` to the Kubernetes system.

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

10. Create the "JCNR ConfigMap" on page 134 if using the Virtual Router Redundancy Protocol (VRRP) for your JCNR cluster. Apply the `jcnr-aws-config.yaml` to the Kubernetes system.

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

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

    See, "Customize JCNR Helm Chart for EKS Deployment" on page 137 for descriptions of the helm chart configurations and a sample helm chart for EKS deployment.

12. Optionally, customize JCNR configuration.

    See, "Customize JCNR Configuration " on page 54 for creating and applying the cRPD customizations.

13. Install the Amazon EBS CSI driver.

14. Label the nodes to which JCNR must be installed based on the `nodeAffinity` defined in the `values.yaml`. For example:

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

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

    Run the following command:

```
helm install jcnr .
```

```
NAME: jcnr
LAST DEPLOYED: Fri Sep 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               2023-09-22 06:04:33.144611017 -0400 EDT
deployed        jcnr-23.3.0               23.3.0
```

Verify JCNR Installation on Amazon EKS

1. Verify the state of the JCNR 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          contrail-vrouter-nodes-wmr26            3/3     Running   0
13d
jcnr              kube-crpd-worker-sts-3                  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 JCNR daemonsets by issuing the `kubectl get ds -A` command. Use the `kubectl get ds -A` command to get a list of daemonsets. The JCNR daemonsets are highlighted in **bold** text.

```
kubectl get ds -A
```

```
NAMESPACE     NAME                       DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR              AGE
contrail      contrail-vrouter-masters   0         0         0       0            0
<none>                     13d
contrail      contrail-vrouter-nodes     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 JCNR statefulsets by issuing the `kubectl get statefulsets -A` command. The command output provides the statefulsets.

```
kubectl get statefulsets -A
```

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

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

   a. View the *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 JCNR 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 JCNR 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

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

Table 11: Cloud-Native Router Minimum Host System Requirements

Component	Value/Version
EKS Deployment	Self-managed Nodes
Host OS	Amazon Linux 2
EKS version	1.25.12
Instance Type	Any instance type with ena adapters
Kernel Version	The tested kernel version is 5.15.0-1040-aws

Table 11: Cloud-Native Router Minimum Host System Requirements *(Continued)*

Component	Value/Version
NIC	Elastic Network Adapter (ENA)
Kubernetes (K8s)	1.26.3, 1.28.x
AWS CLI version	2.11.9
VPC CNI	v1.14.0-eksbuild.3
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	containterd

Resource Requirements

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

Table 12: Cloud-Native Router Resource Requirements

Resource	Value	Usage Notes
Data plane forwarding cores	2 cores (2P + 2S)	
Service/Control Cores	0	

Table 12: Cloud-Native Router Resource Requirements *(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`
		Enable Unsafe IOMMU mode
		`echo Y > /sys/module/vfio_iommu_type1/parameter/allow_unsafe_interrupts`
		`echo Y > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode`

Hugepages (1G)	6 Gi	Add `GRUB_CMDLINE_LINUX_DEFAULT` values in **/etc/default/grub** on the host. For example: `GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=8 intel_iommu=on iommu=pt"`
		Update grub and reboot the host. For example:
		`grub2-mkconfig -o /boot/grub2/grub.cfg`
		Verify the hugepage is set by executing the following commands:
		`cat /proc/cmdline`
		`grep -i hugepages /proc/meminfo`

JCNR Controller cores	.5	
JCNR vRouter Agent cores	.5	

Miscellaneous Requirements

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

Table 13: Miscellaneous Requirements

Cloud-Native Router Release Miscellaneous Requirements
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 the `AmazonEBSCSIDriverPolicy` IAM policy to the role assigned to the EKS cluster.

Set IOMMU and IOMMU-PT in `/etc/default/grub` file. For example:
`GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=8 intel_iommu=on iommu=pt"`
Update grub and reboot the host. For example:
`grub2-mkconfig -o /boot/grub2/grub.cfg`

Additional kernel modules need to be loaded on the host before deploying JCNR in L3 mode. These modules are usually available in `linux-modules-extra` or `kernel-modules-extra` packages. Add each of following kernel modules on a separate line to `/etc/modules-load.d/crpd.conf` to load the modules at boot:
`cat /etc/modules-load.d/crpd.conf`
`tun`
`fou`
`fou6`
`ipip`
`ip_tunnel`
`ip6_tunnel`
`mpls_gso`
`mpls_router`
`mpls_iptunnel`
`vrf`
`vxlan`
NOTE: Applicable for L3 deployments only.

Table 13: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

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

JCNR ConfigMap for VRRP

You can enable Virtual Router Redundancy Protocol (VRRP) for your JCNR cluster.

You must create a JCNR ConfigMap to define the behavior of VRRP for your JCNR 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 JCNR to specific routing tables in AWS. An example `jcnr-aws-config.yaml` manifest is provided:

```
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"
      }
    ]
```

The table provided below describes the ConfigMap elements:

Table 14: JCNR ConfigMap Elements

Element	Description
jcnr-table-name	The routing table in JCNR from which prefixes should be copied.
jcnr-policy-name	A routing policy in JCNR that imports the prefixes in the named routing table to copy to the AWS routing table.
jcnr-nexthop-interface-name	Name of the JCNR interface which should be used as the next-hop by the AWS routing table when this instance of the JCNR is VRRP master.
vpc-table-tag	A freeform tag applied to the routing table in AWS to which the prefixes should be copied.

The jcnr-aws-config.yaml must be applied to the Kubernetes system before JCNR installation. The JCNR CNI deployer renders the cRPD configuration based on the ConfigMap.

NOTE: When not using VRRP, you must 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 15: Cloud-Native Router Listening Ports

Protocol	Port	Description
TCP	8085	vRouter introspect–Used to gain internal statistical information about vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from JCNR control plane
TCP	9091	vRouter health check–cloud-native router checks to ensure **contrail-vrouter-dpdk** process is running, etc.
TCP	50052	gRPC port–JCNR listens on both IPv4 and IPv6
TCP	8081	JCNR Deployer Port
TCP	22	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	50051	**jsd** on cRPD
TCP	51051	**jsd** on cRPD
UDP	50055	Syslog-NG

Download Options

To deploy JCNR on an EKS cluster you can either download the helm charts from the Juniper Support Site or subscribe via the AWS Marketplace.

NOTE:
`https://enterprise.hub.juniper.net`

JCNR Licensing

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

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

Customize JCNR 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 Attributes and Descriptions

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.

Table 16: Helm Chart Attributes and Descriptions

Key	Additional Key Configuration	Description
registry		Defines the docker registry for the vRouter, cRPD and jcnr-cni container images. The default value is set to: • Juniper Enterprise Hub for helm charts downloaded from the Juniper Support Site. • Amazon Elastic Container Registry (ECR) for helm charts downloaded from the AWS marketplace.
repository		Defines the repository path for the vRouter, cRPD and jcnr-cni container images. This is a global key and takes precedence over "repository" paths under "common" section. The default value is: • `jcnr-container-prod/` for Juniper Support package. • `juniper-networks` for AWS Marketplace subscriptions.
imagePullSecret		Defines the registry authentication credentials. View the "Configure Repository Credentials" on page 301 topic for more information.
	registryCredentials	Base64 representation of your Docker registry credentials.
	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 default.

Table 16: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
	repository	Defines the repository path. The global repository key takes precedence if defined. The default value is set to: • `atom-docker/cn2/bazel-build/dev/` for Juniper Support package. • `juniper-networks` for AWS Marketplace subscriptions.
	tag	Defines the image tag. The default value is configured to the appropriate tag number for the JCNR release version.
replicas		(Optional) Indicates the number of cRPD replicas deployed on the worker nodes in a multi-node cluster. If the value is not specified, the default value 1 is considered. The value for this key must be specified for multi-node clusters and must match the number of nodes to which JCNR must be deployed.
storageClass		Indicates the name of the storage class for cRPD. Must be specified as `gp2` for EKS deployments.
awsregion		Defines the AWS region for the EKS deployment.
noLocalSwitching		Not applicable for EKS deployments.

Table 16: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description

fabricInterface		Provide a list of interfaces to be bound to the DPDK. You can also provide subnets instead of interface names. If both the interface name and the subnet are specified, then the interface name takes precedence over subnet/gateway combination. The subnet/gateway combination is useful when the interface names vary in a multi-node cluster.
		
		NOTE: 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:
		
		`# L3 only`
		`- eth1:`
		`ddp: "off"`
		`- eth2:`
		`ddp: "off"`

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

	ddp	Not applicable for EKS deployments.
	interface_mode	Not applicable for EKS deployments.
	vlan-id-list	Not applicable for EKS deployments.

Table 16: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
	storm-control-profile	Not applicable for EKS deployments.
	native-vlan-id	Not applicable for EKS deployments.
	no-local-switching	Not applicable for EKS deployments.
fabricWorkloadInterface		Not applicable for EKS deployments.
log_level		Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave the log_level set to the default INFO unless instructed to change it by Juniper support.
log_path		The defined directory stores various JCNR related descriptive logs such as contrail-vrouter-agent.log, contrail-vrouter-dpdk.log, etc. The default value is /var/log/jcnr/.
syslog_notifications		Indicates the absolute path to the file that stores syslog-ng generated notifications in JSON format. The default value is /var/log/jcnr/jcnr_notifications.json.
corePattern		Indicates the core pattern to denote how the core file is generated. If this configuration is left blank, then JCNR pods will not overwrite the default pattern. **NOTE**: Set the corePattern value on host before deploying JCNR. You may change the value in `/etc/sysctl.conf`. For example, `kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`
coreFilePath		Indicates the path for the core file. If the value is left blank, then vRouter considers /var/crashes as the default value.

Table 16: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description

nodeAffinity		(Optional) Defines labels on nodes to determine where to place the vRouter pods. By default the vRouter pods are deployed to all worker 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 JCNR must 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 cni path with a path used by your distribution e.g. /var/opt/cni/bin.
grpcTelemetryPort		(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50051.
grpcVrouterPort		(Optional) Enter a value for this parameter to override vRouter gRPC server default port of 50052.
vRouterDeployerPort		(Optional) Default value is 8081. Configure to override if the default port is unavailable.

Table 16: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
restoreInterfaces		Recommend to set the value of this key to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts.
bondInterfaceConfigs		Not applicable for EKS deployments.
mtu		Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default value is 9000.
cpu_core_mask		Indicates the vRouter forward core mask. If qos is enabled, you will need to allocate 4 CPU cores. Use the cores not used by the host OS in your EC2 instance.
stormControlProfiles		Not applicable for EKS deployments.
dpdkCommandAdditionalArgs		Pass any additional dpdk cmd line parameters. The --yield_option 0 is set by default and it implies the dpdk forwarding cores will not yield the cpu cores it is assigned to. Additional 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"`
ddp		Not applicable for EKS deployments.
qosEnable		Set to false for EKS deployments.
vrouter_dpdk_uio_driver		The uio driver is `vfio-pci`.
agentModeType		Can be dpdk or xdp. Setting agentModeType to dpdk will bringup dpdk datapath. Setting agentModeType to xdp uses ebpf. The default value is dpdk.

Table 16: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
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.
persistConfig		Set this flag to true if you wish jcnr-cni generated pod configuration to persist even after uninstallation. The option must be set only for L2 mode. The default value is false.

NOTE: For Amazon EKS, you need to additionally update the `dpdkCommandAdditionalArgs` key and set `tx` and `rx` descriptors to 256. For example:

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

Sample Helm Charts

Sample EKS JCNR helm charts are provided below:

Helm Chart for Amazon EKS Deployment (Subscription via Juniper Support Site)

A working Amazon EKS L3 helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

```
  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred

 common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

  # 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: “2”

  # storageClass: Name of the storage class for cRPD. This option is must for
  # cloud deployments such as AWS where gp2 can be used
  storageClass: gp2
  # Set AWS Region for AWS deployments
  awsregion: us-east-1
  #noLocalSwitching: [700]

  # fabricInterface: provide a list of interfaces to be bound to dpdk
  # You can also provide subnets instead of interface names. Interfaces name take precedence over
  # Subnet/Gateway combination if both specified (although there is no reason to specify both)
  # Subnet/Gateway combination comes handy when the interface names vary in a multi-node cluster
  fabricInterface:
  ##########################
  # L2 only
  #- eth1:
  #    ddp: "auto"               # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
```

```
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  #- eth2:
  #    ddp: "auto"               # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [700]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  #- bond0:
  #    ddp: "auto" # auto/on/off  # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    #native-vlan-id: 100
  #    #no-local-switching: true

  ##########################
  # L3  only
  #- eth11:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  ########################

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

```
  ###################################

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
    - subnet: 10.0.3.0/24
      gateway: 10.0.3.1
      ddp: "off"
    - subnet: 10.0.5.0/24
      gateway: 10.0.5.1
      ddp: "off"

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

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

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

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

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

```
  # 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: key1
    operator: In
    values:
    - jcnr

  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  #cni_bin_dir:  /var/lib/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
  #vRouterDeployerPort: 8082


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

 # Enable bond interface configurations L2 only or L2 L3 deployment

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

  # MTU for all physical interfaces( all VF's and  PF's)
```

```
  mtu: "9000"

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Helm Chart for EKS Deployment (Subscription via AWS Marketplace)

A working Amazon EKS L3 helm chart sample is shown below. The configured sections are highlighted in **bold**:

```
#####################################################################
#                 Common Configuration (global vars)                #
#####################################################################
global:
  registry: 709825985650.dkr.ecr.us-east-1.amazonaws.com/
  # uncomment below if all images are available in the same path; it will
  # take precedence over "repository" paths under "common" section below
  repository: juniper-networks/

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred

  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

  # 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: “2”

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

```
  storageClass: gp2
  # Set AWS Region for AWS deployments
  awsregion: us-east-1
  #noLocalSwitching: [700]

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

  ##########################
  # L3  only
  #- eth11:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
```

```
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off

  ########################

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

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
    - subnet: 10.0.3.0/24
      gateway: 10.0.3.1
      ddp: "off"
    - subnet: 10.0.5.0/24
      gateway: 10.0.5.1
      ddp: "off"

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

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

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

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

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

  # 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: key1
    operator: In
    values:
    - jcnr

  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  #cni_bin_dir:  /var/lib/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
  #vRouterDeployerPort: 8082
```

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

 # Enable bond interface configurations L2 only or L2 L3 deployment

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

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

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Customize JCNR Configuration

SUMMARY

Read this topic to understand how to customize JCNR configuration using a ConfigMap.

IN THIS SECTION

JCNR ConfigMap

Starting with Juniper Cloud-Native Router (JCNR) Release 23.3, JCNR supports customizing configuration using a ConfigMap when deployed in L3 mode. In cloud-based deployments, in the event of a node failure, the JCNR pods may be spawned on newer or different nodes. A ConfigMap decouples the configuration parameters from node names and is based on node labels instead. This enables the JCNR CNI deployer to consume the configuration parameters, apply them to the cRPD configuration template and render the configuration, as long as a matching label is available for the node.

A ConfigMap is an API object to store data in key-values pairs. A ConfigMap defines per node variables that are consumed by nodes matching the label. The key-value pairs are used to render the configuration via a go template. The configured template must be available in the `Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/` directory for the configuration to be applied to the cRPD pods.

NOTE: You must apply the ConfigMap before installing JCNR to create cRPD pods with custom configuration. The cRPD pod must be deleted and respawned should you wish to apply the configuration parameters any time after JCNR installation. The configuration parameters are applied by default to any newly spawned cRPD pods. The JCNR customization via ConfigMap is optional.

NOTE: JCNR also supports customization via node annotations for backward compatibility with previous releases. Considering that node annotations are coupled with node names, it is highly recommended to customize JCNR via ConfigMaps, specifically for cloud deployments. Refer to Customize JCNR Configuration using node annotations for more information.

Configuration Example

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

You define the key-value pair for different node labels in your cluster. An example of the `jcnr-params-configmap.yaml` file is provided below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: jcnr-params
  namespace: jcnr
data:
  jcnr1: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0001.00",
      "IPv4LoopbackAddr": "110.1.1.2",
      "srIPv4NodeIndex": "2000",
      "srIPv6NodeIndex": "3000",
```

```
      "BGPIPv4Neighbor": "110.1.1.254",
      "BGPLocalAsn": "64512"
    }
  jcnr2: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0000.00",
      "IPv4LoopbackAddr": "110.1.1.3",
      "srIPv4NodeIndex": "2001",
      "srIPv6NodeIndex": "3001",
      "BGPIPv4Neighbor": "110.1.2.254",
      "BGPLocalAsn": "64512"
    }
```

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

```
apply-groups [custom];
groups {
    custom {
        interfaces {
            lo0 {
                unit 0 {
                {{if .Params.isoLoopbackAddr}}
                    family iso {
                        address {{.Params.isoLoopbackAddr}};
                    }
                {{end}}
                    family inet {
                        address {{.Params.IPv4LoopbackAddr}};
                    }
                }
            }
        }
        routing-options {
            router-id {{.Params.IPv4LoopbackAddr}}
            route-distinguisher-id {{.Params.IPv4LoopbackAddr}}
        }
        protocols {
            isis {
                interface all;
                {{if and .Env.SRGB_START_LABEL .Env.SRGB_INDEX_RANGE}}
                source-packet-routing {
```

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

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

NOTE: You can define additional cRPD configuration hierarchies in the template. The values to be rendered from the ConfigMap defined in the `jcnr-params-configmap.yaml` must be defined as {{.Params.*var-name*}}. Any environment variables, such as variables defined in `values.yaml`, must be defined as {{.Env.*variable_name*}}.

Complete the following steps to apply the customizations.

1. Label each node based on the keys used in the ConfigMap.

```
kubectl label nodes <node_name1> jcnr.juniper.net/params-profile=jcnr1
kubectl label nodes <node_name2> jcnr.juniper.net/params-profile=jcnr2
```

2. Apply the ConfigMap to the cluster nodes using the command provided below:

```
# kubectl apply -f jcnr-params-configmap.yaml
configmap/jcnr-params created
```

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

```
# cp Juniper_Cloud_Native_Router_release_number/helmchart/cRPD_examples/jcnr-cni-custom-
config-cm.tmpl Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/
#
```

4. Deploy the cloud-native router components, including the cRPD. Once the installation completes, *access the cRPD CLI* and issue the `show configuration | display set` command in the `cli` mode to view the custom configuration you applied.

```
root@jcnr-01> show configuration
## Last commit: 2023-06-23 08:30:42 EDT by root
version 20230608.143922_builder.r1342735;
groups {
    base { /* OMITTED */ };
    custom {
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 110.1.1.2/32;
                    }
                    family iso {
                        address 49.0004.1000.0000.0001.00;
                    }
                }
            }
        }
        policy-options {
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then {
                    community add udp;
                }
            }
            community udp members encapsulation:0L:13;
        }
        routing-options {
            route-distinguisher-id 110.1.1.2;
            router-id 110.1.1.2;
            dynamic-tunnels {
                dyn-tunnels {
                    source-address 110.1.1.2;
                    udp;
                    destination-networks {
                        110.1.1.254/32;
                    }
                }
```

```
            }
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address 110.1.1.2;
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                    local-as 64512;
                    neighbor 110.1.1.254;
                }
            }
            isis {
                interface all;
                source-packet-routing {
                    srgb start-label 400000 index-range 4000;
                    node-segment {
                        ipv4-index 2000;
                        ipv6-index 3000;
                    }
                }
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
    }
    cni { /* OMITTED */ };
    internal { /* OMITTED */ };
}
apply-groups [ custom base internal ];
```

Modifying the ConfigMap

If you wish to change the ConfigMap any time after JCNR installation, you must delete the cRPD pod and respawn it using the following command:

```
kubectl patch sts kube-crpd-worker-sts -n jcnr -p '{"spec":{"template":{"metadata":
{"annotations":{"configmap-reload/timestamp":"'$(date +%s)'"}}}}}'
```

This triggers a rolling restart of all cRPD pods. Alternatively, you can identify the cRPD pods on nodes for which the ConfigMap has changed and manually delete the pod. The ConfigMap changes will be applied automatically to any respawned pods.

Troubleshooting

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

5

CHAPTER

Install Cloud-Native Router on Google Cloud Platform

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 173 section to ensure the setup has all the required configuration.
2. Download the JCNR helm charts, **Juniper_Cloud_Native_Router_*release-number*.tgz**, to the directory of your choice. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

```
tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
```

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

```
cd Juniper_Cloud_Native_Router_release-number
```

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

5. View the contents in the current directory.

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

6. Enter the root password for your host server and your Juniper Cloud-Native Router license file into the **secrets/jcnr-secrets.yaml** file. You must enter the password and license in base64 encoded format.

   You can view the sample contents of the **jcnr-secrets.yaml** file below:

```
---
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>
```

   To encode the password, create a file with the plain text password on a single line. Then issue the command:

```
base64 -w 0 rootPasswordFile
```

   To encode the license, copy the license key into a file on your host server and issue the command:

```
base64 -w 0 licenseFile
```

You must copy the base64 outputs and paste them into the **secrets/jcnr-secrets.yaml** file in the appropriate locations.

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

Apply the **secrets/jcnr-secrets.yaml** to the Kubernetes system.

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

> **NOTE**: Starting with JCNR Release 23.2, the JCNR license format has changed. Request a new license key from the JAL portal before deploying or upgrading to 23.2 or newer releases.

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

   See, "Customize JCNR Helm Chart for GCP Deployment" on page 183 for descriptions of the helm chart configurations and a sample helm chart for GCP deployment..

8. Optionally, customize JCNR configuration.

   See, "Customize JCNR Configuration " on page 54 for creating and applying the cRPD customizations.

9. Label the nodes to which JCNR must be installed based on the `nodeaffinity` defined in the `values.yaml`. For example:

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

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

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

```
helm install jcnr .
```

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

11. 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-23.3.0               23.3.0
```

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.

(Continued)

Settings	Value
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.
JCNR 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 JCNR license field.
cRPD Config Template	Create a config template to customize JCNR configuration. See, "Customize JCNR Configuration (Google Cloud Marketplace)" on page 202 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 JCNR configuration. See, "Customize JCNR Configuration (Google Cloud Marketplace)" on page 202 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

(Continued)

Settings	Value
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 173 section for additional minimum system requirements. Please note that the settings are pre-configured for the JCNR deployment via Google Cloud Marketplace.
4. Click `Deploy` to complete the JCNR deployment.
5. Once deployed, you can customize the JCNR helm chart. Review the "Customize JCNR Helm Chart for GCP Deployment" on page 183 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 JCNR deployment.

1. Verify the state of the JCNR pods by issuing the `kubectl get pods -A` command.

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

```
kubectl get pods -A
```

```
NAMESPACE         NAME                                      READY   STATUS
RESTARTS        AGE
```

```
contrail-deploy   contrail-k8s-deployer-579cd5bc74-g27gs    1/1     Running
0                 103s
contrail          contrail-vrouter-masters-lqjqk            3/3     Running
0                 87s
jcnr              kube-crpd-worker-sts-0                    1/1     Running
0                 103s
jcnr              syslog-ng-ds5qd                           1/1     Running
0                 103s
kube-system       calico-kube-controllers-5f4fd8666-m78hk   1/1     Running
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 JCNR daemonsets by issuing the `kubectl get ds -A` command.

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

```
kubectl get ds -A
```

```
NAMESPACE     NAME                       DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR              AGE
contrail      contrail-vrouter-masters   1         1         1       1            1
<none>                     90m
contrail      contrail-vrouter-nodes     0         0         0       0            0
<none>                     90m
```

```
jcnr          syslog-ng                1         1         1         1           1
<none>                     90m
kube-system   calico-node              1         1         1         1           1
kubernetes.io/os=linux     86d
kube-system   kube-multus-ds-amd64     1         1         1         1           1
kubernetes.io/arch=amd64   86d
kube-system   kube-proxy               1         1         1         1           1
kubernetes.io/os=linux     86d
kube-system   nodelocaldns             1         1         1         1           1
kubernetes.io/os=linux     86d
```

3. Verify the JCNR statefulsets by issuing the `kubectl get statefulsets -A` command.

   The command output provides the statefulsets.

```
kubectl get statefulsets -A
```

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

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

   a. View the *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 JCNR 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 JCNR 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 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

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

NOTE: The settings below are pre-configured when you deploy JCNR via the Google Cloud Marketplace.

Table 17: Cloud-Native Router Minimum Host System Requirements

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	The tested kernel version is 4.18.0-477.15.1.el8_8.cloud.x86_64
NIC	VirtIO NIC	

Table 17: Cloud-Native Router Minimum Host System Requirements *(Continued)*

Component	Value/Version	Notes
Kubernetes (K8s)	Version 1.25.x	The tested K8s version is 1.25.5. The K8s version for Google Cloud Marketplace JCNR subscription is v1.27.5.
Calico	Version 3.25.1	
Multus	Version 4.0	
Helm	3.9.x	
Container-RT	containerd	

Resource Requirements

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

Table 18: Cloud-Native Router Resource Requirements

Resource	Value	Usage Notes
Data plane forwarding cores	2 cores	
Service/Control Cores	0	

Table 18: Cloud-Native Router Resource Requirements *(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`
		Enable Unsafe IOMMU mode
		`echo Y > /sys/module/vfio_iommu_type1/parameter/allow_unsafe_interrupts`
		`echo Y > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode`

Hugepages (1G)	6 Gi	Add `GRUB_CMDLINE_LINUX_DEFAULT` values in **/etc/default/grub** and reboot the host. For example:
		`GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=64 intel_iommu=on iommu=pt"`
		Update grub and reboot the host. For example:
		`grub2-mkconfig -o /boot/grub2/grub.cfg`
		Verify the hugepage is set by executing the following commands:
		`cat /proc/cmdline`
		`grep -i hugepages /proc/meminfo`

JCNR Controller cores	.5	

Table 18: Cloud-Native Router Resource Requirements *(Continued)*

Resource	Value	Usage Notes
JCNR vRouter Agent cores	.5	

Miscellaneous Requirements

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

Table 19: Miscellaneous Requirements

Cloud-Native Router Release Miscellaneous Requirements

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

Table 19: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

Additional kernel modules need to be loaded on the host before deploying JCNR in L3 mode. These modules are usually available in `linux-modules-extra` or `kernel-modules-extra` packages. Run the following commands to add the kernel modules:
 	
`cat /etc/modules-load.d/crpd.conf`
`tun`
`fou`
`fou6`
`ipip`
`ip_tunnel`
`ip6_tunnel`
`mpls_gso`
`mpls_router`
`mpls_iptunnel`
`vrf`
`vxlan`
 	
NOTE: Applicable for L3 deployments only.

Run the `ip fou add port 6635 ipproto 137` command on the Linux host to enable kernel based forwarding.

Enable IP Forwarding for VMs in GCP. Use one of the two methods to enable it:
1. Specify it as an option while creating the VM. For example:
 	
`gcloud compute instances create instance-name --can-ip-forward`
 	
2. For an exisiting VM, enable IP forwarding by updating the compute instance via a file. For example:
 	
`gcloud compute instances export transit-jcnr01 --project jcnr-ci-admin --zone us-west1-a --destination=instance_file_1`
 	
Edit the instance file to set the value `canIpForward=true`.
 	
Update the compute instance from the file:
 	
`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`

Table 19: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

Enable Multi-IP subnet on Guest OS:
`gcloud compute images create debian-9-multi-ip-subnet \`
`--source-disk debian-9-disk \`
`--source-disk-zone us-west1-a \`
`--guest-os-features MULTI_IP_SUBNET`

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 JCNR 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, 2.51.2.0/23, 2.51.1.0/24, 2.2.2.2/32, 3.3.3.3/32 7. `Protocols`: all 8. `Enforcement`: Enabled where 10.2.0.0/24 is the subnet to which JCNR is attached and 2.51.2.0/24, 2.51.1.0/24, 2.2.2.2/32, 3.3.3.3/32 are loopback IP ranges.
JCNR supports only IPv4 for GCP.
JCNR deployment on GCP supports only N8-standard for VM deployments. The N16-standard is not supported.

Table 19: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

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

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 20: Cloud-Native Router Listening Ports

Protocol	Port	Description
TCP	8085	vRouter introspect–Used to gain internal statistical information about vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from JCNR control plane
TCP	9091	vRouter health check–cloud-native router checks to ensure **contrail-vrouter-dpdk** process is running, etc.
TCP	50052	gRPC port–JCNR listens on both IPv4 and IPv6
TCP	8081	JCNR Deployer Port
TCP	22	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	50051	**jsd** on cRPD
TCP	51051	**jsd** on cRPD

Table 20: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
UDP	50055	Syslog-NG

Download Options

To deploy JCNR on GCP you can either download the helm charts from the Juniper Support Site or subscribe via the Google Cloud Marketplace.

NOTE:
`https://enterprise.hub.juniper.net`

JCNR Licensing

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

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

Customize JCNR 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 Attributes and Descriptions

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

Table 21: Helm Chart Attributes and Descriptions

Key	Additional Key Configuration	Description
registry		Defines the docker registry where the vRouter, cRPD and jcnr-cni container images are hosted. The default value is set to Juniper Enterprise Hub.
repository		Defines the repository path for the vRouter, cRPD and jcnr-cni container images. This is a global key and takes precedence over "repository" paths under "common" section. The default value is `jcnr-container-prod/`

Table 21: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
imagePullSecret		(Optional) Defines the registry authentication credentials. You can configure credentials to either the Juniper repository or your private registry. View the "Configure Repository Credentials" on page 301 topic for more information.
	registryCredentials	Base64 representation of your Docker registry credentials.
	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.
	repository	Defines the repository path. The global repository key takes precedence if defined. The default value is set to `atom-docker/cn2/bazel-build/dev/`.
	tag	Defines the image tag. The default value is configured to the appropriate tag number for the JCNR release version.
replicas		(Optional) Indicates the number of replicas for cRPD. If the value is not specified, then the default value 1 is considered. The value for this key must be specified for multi-node clusters and must match the number of nodes to which JCNR must be deployed.
storageClass		Not applicable for GCP deployments.
awsregion		Not applicable for GCP deployments.
noLocalSwitching		Not applicable for GCP deployments.

Table 21: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description

fabricInterface		Provide a list of interfaces to be bound to the DPDK. You can also provide subnets instead of interface names. If both the interface name and the subnet are specified, then the interface name takes precedence over subnet/gateway combination. The subnet/gateway combination is useful when the interface names vary in a multi-node cluster.
		
		NOTE: Use the L3 only section to configure fabric interfaces for GCP. The L2 only and L2-L3 sections are not applicable for GCP deployments.
		
		For example:
		
		`# L3 only`
		`- eth1:`
		`ddp: "off"`
		`- eth2:`
		`ddp: "off"`

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

	ddp	Not applicable for GCP deployments.
	interface_mode	Not applicable for GCP deployments.
	vlan-id-list	Not applicable for GCP deployments.

Table 21: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
	storm-control-profile	Not applicable for GCP deployments.
	native-vlan-id	Not applicable for GCP deployments.
	no-local-switching	Not applicable for GCP deployments.
fabricWorkloadInterface		Not applicable for GCP deployments.
log_level		Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave the log_level set to INFO unless instructed to change it by Juniper support.
log_path		The defined directory stores various JCNR related descriptive logs such as contrail-vrouter-agent.log, contrail-vrouter-dpdk.log, etc.
syslog_notifications		Indicates the absolute path to the file that stores syslog-ng generated notifications in JSON format.
corePattern		Indicates the core pattern to denote how the core file is generated. If this configuration is left blank, then JCNR pods will not overwrite the default pattern. **NOTE**: Set the corePattern value on host before deploying JCNR. You may change the value in `/etc/sysctl.conf`. For example, `kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`
coreFilePath		Indicates the path for the core file. If the value is left blank, then vRouter considers /var/crashes as the default value.

Table 21: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description

nodeAffinity		(Optional) Defines labels on nodes to determine where to place the vRouter pods. By default the vRouter pods are deployed to all worker 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 JCNR must 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 cni path with a path of used by your distribution e.g. /var/opt/cni/bin.
grpcTelemetryPort		(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50051.
grpcVrouterPort		(Optional) Enter a value for this parameter to override vRouter gRPC server default port of 50052.
restoreInterfaces		Set the value of this key to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts.
vRouterDeployerPort		(Optional) Default value is 8081. Configure to override if the default port is unavailable.

Table 21: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
bondInterfaceConfigs		Not applicable for GCP deployments.
mtu		Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default value is 9000.
cpu_core_mask		Indicates the vRouter forward core mask. If qos is enabled, you will need to allocate 4 CPU cores (primary and siblings).
stormControlProfiles		Not applicable for GCP deployments.
dpdkCommandAdditionalArgs		Pass any additional dpdk cmd line parameters. The --yield_option 0 is set by default and it implies the dpdk forwarding cores will not yield the cpu cores it is assigned to. Additional 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"`
ddp		Not applicable for GCP deployments.
qosEnable		Set to false for GCP deployments.
vrouter_dpdk_uio_driver		The uio driver is `vfio-pci`.
agentModeType		Can be dpdk or xdp. Setting agentModeType to dpdk will bringup dpdk datapath. Setting agentModeType to xdp uses ebpf. The default value is 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.

Table 21: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
persistConfig		Set this flag to true if you wish jcnr-cni generated pod configuration to persist even after uninstallation. The option must be set only for L2 mode. The default value is false.

NOTE: If you are installing JCNR on GCP, then update the `dpdkCommandAdditionalArgs` key and set `tx` and `rx` descriptors to 256. For example:

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

Sample Helm Charts

Sample GCP JCNR helm chart is provided below:

Helm Chart for GCP Deployment

A working GCP L3 helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
```

```
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred

  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

  # 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: “2”

  # storageClass: Name of the storage class for cRPD. This option is must for
  # cloud deployments such as AWS where gp2 can be used
  # storageClass: gp2
  # Set AWS Region for AWS deployments
  # awsregion: us-east-1
  #noLocalSwitching: [700]

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

```
  #    no-local-switching: true
  #- eth2:
  #    ddp: "auto"               # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [700]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  #- bond0:
  #    ddp: "auto" # auto/on/off  # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    #native-vlan-id: 100
  #    #no-local-switching: true

  ##########################
  # L3  only
  #- eth11:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  #- eth2:
  #    ddp: "off"                # ddp parameter is optional; options include auto or on or
off; default: off
  ########################

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

  # Provide subnets instead of interface names
```

```
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
  # - subnet: 10.0.3.0/24
  #   gateway: 10.0.3.1
  #   ddp: "off"
  # - subnet: 10.0.5.0/24
  #   gateway: 10.0.5.1
  #   ddp: "off"

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

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

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

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

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

  # 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: key1
  #  operator: In
  #  values:
  #  - jcnr

  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  #cni_bin_dir:  /var/lib/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
  #vRouterDeployerPort: 8082


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

 # Enable bond interface configurations L2 only or L2 L3 deployment

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

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

  # vrouter fwd core mask
```

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Customize JCNR Configuration

SUMMARY

Read this topic to understand how to customize JCNR configuration using a ConfigMap.

IN THIS SECTION

JCNR ConfigMap

Starting with Juniper Cloud-Native Router (JCNR) Release 23.3, JCNR supports customizing configuration using a ConfigMap when deployed in L3 mode. In cloud-based deployments, in the event of a node failure, the JCNR pods may be spawned on newer or different nodes. A ConfigMap decouples the configuration parameters from node names and is based on node labels instead. This enables the JCNR CNI deployer to consume the configuration parameters, apply them to the cRPD configuration template and render the configuration, as long as a matching label is available for the node.

A ConfigMap is an API object to store data in key-values pairs. A ConfigMap defines per node variables that are consumed by nodes matching the label. The key-value pairs are used to render the configuration via a go template. The configured template must be available in the `Juniper_Cloud_Native_Router_`*`release_number`*`/helmchart/charts/jcnr-cni/files/` directory for the configuration to be applied to the cRPD pods.

NOTE: You must apply the ConfigMap before installing JCNR to create cRPD pods with custom configuration. The cRPD pod must be deleted and respawned should you wish to apply the configuration parameters any time after JCNR installation. The configuration parameters are applied by default to any newly spawned cRPD pods. The JCNR customization via ConfigMap is optional.

NOTE: JCNR also supports customization via node annotations for backward compatibility with previous releases. Considering that node annotations are coupled with node names, it is highly recommended to customize JCNR via ConfigMaps, specifically for cloud deployments. Refer to Customize JCNR Configuration using node annotations for more information.

Configuration Example

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

You define the key-value pair for different node labels in your cluster. An example of the `jcnr-params-configmap.yaml` file is provided below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: jcnr-params
  namespace: jcnr
data:
  jcnr1: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0001.00",
      "IPv4LoopbackAddr": "110.1.1.2",
      "srIPv4NodeIndex": "2000",
      "srIPv6NodeIndex": "3000",
      "BGPIPv4Neighbor": "110.1.1.254",
      "BGPLocalAsn": "64512"
    }
  jcnr2: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0000.00",
      "IPv4LoopbackAddr": "110.1.1.3",
      "srIPv4NodeIndex": "2001",
      "srIPv6NodeIndex": "3001",
      "BGPIPv4Neighbor": "110.1.2.254",
      "BGPLocalAsn": "64512"
    }
```

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

```
apply-groups [custom];
groups {
    custom {
        interfaces {
            lo0 {
```

```
                unit 0 {
                {{if .Params.isoLoopbackAddr}}
                    family iso {
                        address {{.Params.isoLoopbackAddr}};
                    }
                {{end}}
                    family inet {
                        address {{.Params.IPv4LoopbackAddr}};
                    }
                }
            }
        }
        routing-options {
            router-id {{.Params.IPv4LoopbackAddr}}
            route-distinguisher-id {{.Params.IPv4LoopbackAddr}}
        }
        protocols {
            isis {
                interface all;
                {{if and .Env.SRGB_START_LABEL .Env.SRGB_INDEX_RANGE}}
                source-packet-routing {
                    srgb start-label {{.Env.SRGB_START_LABEL}} index-range
{{.Env.SRGB_INDEX_RANGE}};
                    node-segment {
                        {{if .Params.srIPv4NodeIndex}}
                        ipv4-index {{.Params.srIPv4NodeIndex}};
                        {{end}}
                        {{if .Params.srIPv6NodeIndex}}
                        ipv6-index {{.Params.srIPv6NodeIndex}};
                        {{end}}
                    }
                }
                {{end}}
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
        policy-options {
```

```
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then community add udp;
            }
            community udp members encapsulation:0L:13;
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address {{.Params.IPv4LoopbackAddr}};
                    local-as {{.Params.BGPLocalAsn}};
                    neighbor {{.Params.BGPIPv4Neighbor}};
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                }
            }
        }
        routing-options {
                dynamic-tunnels {
                dyn-tunnels {
                    source-address {{.Params.IPv4LoopbackAddr}};
                    udp;
                    destination-networks {{.Params.BGPIPv4Neighbor}}/32;
                }
            }
        }
    }
}
```

NOTE: You can define additional cRPD configuration hierarchies in the template. The values to be rendered from the ConfigMap defined in the `jcnr-params-configmap.yaml` must be defined as {{.Params.*var-name*}}. Any environment variables, such as variables defined in `values.yaml`, must be defined as {{.Env.*variable_name*}}.

Complete the following steps to apply the customizations.

1. Label each node based on the keys used in the ConfigMap.

```
kubectl label nodes <node_name1> jcnr.juniper.net/params-profile=jcnr1
kubectl label nodes <node_name2> jcnr.juniper.net/params-profile=jcnr2
```

2. Apply the ConfigMap to the cluster nodes using the command provided below:

```
# kubectl apply -f jcnr-params-configmap.yaml
configmap/jcnr-params created
```

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

```
# cp Juniper_Cloud_Native_Router_release_number/helmchart/cRPD_examples/jcnr-cni-custom-
config-cm.tmpl Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/
#
```

4. Deploy the cloud-native router components, including the cRPD. Once the installation completes, *access the cRPD CLI* and issue the `show configuration | display set` command in the `cli` mode to view the custom configuration you applied.

```
root@jcnr-01> show configuration
## Last commit: 2023-06-23 08:30:42 EDT by root
version 20230608.143922_builder.r1342735;
groups {
    base { /* OMITTED */ };
    custom {
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 110.1.1.2/32;
                    }
                    family iso {
                        address 49.0004.1000.0000.0001.00;
                    }
                }
            }
        }
```

```
policy-options {
    # policy to signal dynamic UDP tunnel attributes to BGP routes
    policy-statement udp-export {
        then {
            community add udp;
        }
    }
    community udp members encapsulation:0L:13;
}
routing-options {
    route-distinguisher-id 110.1.1.2;
    router-id 110.1.1.2;
    dynamic-tunnels {
        dyn-tunnels {
            source-address 110.1.1.2;
            udp;
            destination-networks {
                110.1.1.254/32;
            }
        }
    }
}
protocols {
    bgp {
        group jcnrbgp1 {
            type internal;
            local-address 110.1.1.2;
            family inet-vpn {
                unicast;
            }
            family inet6-vpn {
                unicast;
            }
            local-as 64512;
            neighbor 110.1.1.254;
        }
    }
    isis {
        interface all;
        source-packet-routing {
            srgb start-label 400000 index-range 4000;
            node-segment {
                ipv4-index 2000;
```

```
                        ipv6-index 3000;
                    }
                }
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
    }
    cni { /* OMITTED */ };
    internal { /* OMITTED */ };
}
apply-groups [ custom base internal ];
```

Modifying the ConfigMap

If you wish to change the ConfigMap any time after JCNR installation, you must delete the cRPD pod and respawn it using the following command:

```
kubectl patch sts kube-crpd-worker-sts -n jcnr -p '{"spec":{"template":{"metadata":
{"annotations":{"configmap-reload/timestamp":"'$(date +%s)'"}}}}}'
```

This triggers a rolling restart of all cRPD pods. Alternatively, you can identify the cRPD pods on nodes for which the ConfigMap has changed and manually delete the pod. The ConfigMap changes will be applied automatically to any respawned pods.

Troubleshooting

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

Customize JCNR Configuration (Google Cloud Marketplace)

SUMMARY

Read this topic to understand how to customize JCNR configuration using a ConfigMap.

IN THIS SECTION

JCNR ConfigMap

Starting with Juniper Cloud-Native Router (JCNR) Release 23.3, JCNR supports customizing configuration using a ConfigMap when deployed in L3 mode. In cloud-based deployments, in the event of a node failure, the JCNR pods may be spawned on newer or different nodes. A ConfigMap decouples the configuration parameters from node names and is based on node labels instead. This enables the JCNR CNI deployer to consume the configuration parameters, apply them to the cRPD configuration template and render the configuration, as long as a matching label is available for the node.

A ConfigMap is an API object to store data in key-values pairs. A ConfigMap defines per node variables that are consumed by nodes matching the label. The key-value pairs are used to render the configuration via a go template. The configured template must be uploaded to the JCNR deployment wizard in the Google Cloud Marketplace.

NOTE: You must apply the ConfigMap before installing JCNR to create cRPD pods with default or custom configuration. The cRPD pod must be deleted and respawned should you wish to apply the configuration parameters any time after JCNR installation. The configuration parameters are applied by default to any newly spawned cRPD pods.

Configuration Example

Sample ConfigMap and template files are provided below.

You define the key-value pair for different node labels in your cluster. An example of the `cRPD Config Template` file is provided below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: jcnr-params
  namespace: jcnr
data:
  jcnr1: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0001.00",
      "IPv4LoopbackAddr": "110.1.1.2",
      "srIPv4NodeIndex": "2000",
      "srIPv6NodeIndex": "3000",
      "BGPIPv4Neighbor": "110.1.1.254",
      "BGPLocalAsn": "64512"
    }
  jcnr2: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0000.00",
      "IPv4LoopbackAddr": "110.1.1.3",
      "srIPv4NodeIndex": "2001",
      "srIPv6NodeIndex": "3001",
      "BGPIPv4Neighbor": "110.1.2.254",
      "BGPLocalAsn": "64512"
    }
```

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

```
apply-groups [custom];
groups {
    custom {
        interfaces {
            lo0 {
                unit 0 {
                {{if .Params.isoLoopbackAddr}}
                    family iso {
                        address {{.Params.isoLoopbackAddr}};
                    }
```

```
                {{end}}
                    family inet {
                        address {{.Params.IPv4LoopbackAddr}};
                    }
                }
            }
        }
        routing-options {
            router-id {{.Params.IPv4LoopbackAddr}}
            route-distinguisher-id {{.Params.IPv4LoopbackAddr}}
        }
        protocols {
            isis {
                interface all;
                {{if and .Env.SRGB_START_LABEL .Env.SRGB_INDEX_RANGE}}
                source-packet-routing {
                    srgb start-label {{.Env.SRGB_START_LABEL}} index-range
{{.Env.SRGB_INDEX_RANGE}};
                    node-segment {
                        {{if .Params.srIPv4NodeIndex}}
                        ipv4-index {{.Params.srIPv4NodeIndex}};
                        {{end}}
                        {{if .Params.srIPv6NodeIndex}}
                        ipv6-index {{.Params.srIPv6NodeIndex}};
                        {{end}}
                    }
                }
                {{end}}
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
        policy-options {
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then community add udp;
            }
            community udp members encapsulation:0L:13;
```

```
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address {{.Params.IPv4LoopbackAddr}};
                    local-as {{.Params.BGPLocalAsn}};
                    neighbor {{.Params.BGPIPv4Neighbor}};
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                }
            }
        }
        routing-options {
                dynamic-tunnels {
                dyn-tunnels {
                    source-address {{.Params.IPv4LoopbackAddr}};
                    udp;
                    destination-networks {{.Params.BGPIPv4Neighbor}}/32;
                }
            }
        }
    }
}
```

NOTE: You can define additional cRPD configuration hierarchies in the template. The values to be rendered from the ConfigMap defined in the `jcnr-params-configmap.yaml` must be defined as {{.Params.*var-name*}}. Any environment variables, such as variables defined in `values.yaml`, must be defined as {{.Env.*variable_name*}}.

Deploy the cloud-native router. Once the installation completes, *access the cRPD CLI* and issue the `show configuration | display set` command in the `cli` mode to view the custom configuration you applied.

```
root@jcnr-01> show configuration
## Last commit: 2023-06-23 08:30:42 EDT by root
version 20230608.143922_builder.r1342735;
```

```
groups {
    base { /* OMITTED */ };
    custom {
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 110.1.1.2/32;
                    }
                    family iso {
                        address 49.0004.1000.0000.0001.00;
                    }
                }
            }
        }
        policy-options {
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then {
                    community add udp;
                }
            }
            community udp members encapsulation:0L:13;
        }
        routing-options {
            route-distinguisher-id 110.1.1.2;
            router-id 110.1.1.2;
            dynamic-tunnels {
                dyn-tunnels {
                    source-address 110.1.1.2;
                    udp;
                    destination-networks {
                        110.1.1.254/32;
                    }
                }
            }
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address 110.1.1.2;
                    family inet-vpn {
```

```
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                    local-as 64512;
                    neighbor 110.1.1.254;
                }
            }
            isis {
                interface all;
                source-packet-routing {
                    srgb start-label 400000 index-range 4000;
                    node-segment {
                        ipv4-index 2000;
                        ipv6-index 3000;
                    }
                }
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
    }
    cni { /* OMITTED */ };
    internal { /* OMITTED */ };
}
apply-groups [ custom base internal ];
```

Modifying the ConfigMap

If you wish to change the ConfigMap any time after JCNR installation, you must delete the cRPD pod and respawn it using the following command:

```
kubectl patch sts kube-crpd-worker-sts -n jcnr -p '{"spec":{"template":{"metadata":
{"annotations":{"configmap-reload/timestamp":"'$(date +%s)'"}}}}}'
```

This triggers a rolling restart of all cRPD pods. Alternatively, you can identify the cRPD pods on nodes for which the ConfigMap has changed and manually delete the pod. The ConfigMap changes will be applied automatically to any respawned pods.

Troubleshooting

The cRPD pod continues to restart in `CrashLoopBackOff` state if invalid configuration is rendered and applied via the go template. You can apply the rendered configuration manually to a running cRPD pod to validate the configuration and identify issues. For GCP deployment you can find the rendered config in `/config` directory on the JCNR host as `juniper.conf.master`.

CHAPTER 6

Install Cloud-Native Router on Wind River Cloud Platform

Install and Verify Juniper Cloud-Native Router for Wind River 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 into docker and install the cloud-native router components using Helm charts.

1. Review the "System Requirements for Wind River Deployment" on page 218 section to ensure the server has all the required configuration.
2. Download **Juniper_Cloud_Native_Router_*release-number*.tgz**, to the directory of your choice. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

```
tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
```

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

```
cd Juniper_Cloud_Native_Router_release-number
```

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

5. View the contents in the current directory.

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

6. The JCNR container images are required for deployment. You may choose one of the following options:

   a. Download and deploy images from the Juniper repository—`enterprise-hub.juniper.net`. Review the "Configure Repository Credentials" on page 301 topic for instructions on how to configure repository credentials in the deployment helm chart.

   b. You can upload the JCNR images to a local registry. The images are available in the **Juniper_Cloud_Native_Router_*release-number*/images** directory.

7. Enter the root password for your host server and your Juniper Cloud-Native Router license file into the **secrets/jcnr-secrets.yaml** file. You must enter the password and license in base64 encoded format.

   You can view the sample contents of the **jcnr-secrets.yaml** file below:

```
---
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>
```

To encode the password, create a file with the plain text password on a single line. Then issue the command:

```
base64 -w 0 rootPasswordFile
```

To encode the license, copy the license key into a file on your host server and issue the command:

```
base64 -w 0 licenseFile
```

You must copy the base64 outputs and paste them into the **secrets/jcnr-secrets.yaml** file in the appropriate locations.

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

Apply the **secrets/jcnr-secrets.yaml** to the Kubernetes system.

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

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

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

   See, "Customize JCNR Helm Chart for Wind River Deployment" on page 226 for descriptions of the helm chart configurations.

9. Optionally, customize JCNR configuration.

   See, "Customize JCNR Configuration " on page 54 for creating and applying the cRPD customizations.

10. Label the nodes to which JCNR mut be installed based on the `nodeaffinity`, if defined in the `values.yaml`. For example:

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

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

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

```
helm install jcnr .
```

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

12. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

    Sample output:

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

Verify Installation

This section enables you to confirm a successful JCNR deployment.

1. Verify the state of the JCNR pods by issuing the `kubectl get pods -A` command.

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

```
kubectl get pods -A
```

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

2. Verify the JCNR daemonsets by issuing the `kubectl get ds -A` command.

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

```
kubectl get ds -A
```

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

3. Verify the JCNR statefulsets by issuing the `kubectl get statefulsets -A` command.

   The command output provides the statefulsets.

```
kubectl get statefulsets -A
```

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

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

   a. View the *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 JCNR 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 JCNR 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 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 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
            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 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.

Minimum Host System Requirements

This section lists the host system requirements for installing the cloud-native router on a baremetal server.

Table 22: Cloud-Native Router Minimum Host System Requirements

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	11.3 (Bullseye)
Kernel Version	5.10	5.10.0-6-amd64

Table 22: Cloud-Native Router Minimum Host System Requirements *(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	
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	
Helm	3.9.x	
Container-RT	containerd	

Resource Requirements

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

Table 23: Cloud-Native Router Resource Requirements

Resource	Value	Usage Notes
Data plane forwarding cores	2 cores (2P + 2S)	
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	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 64`
		`system host-memory-modify controller-0 1 -1G 64`
		View the huge pages with the following command:
		`system host-memory-list controller-0`
		Unlock the controller:
		`system host-unlock controller-0`

JCNR Controller cores	.5	

Table 23: Cloud-Native Router Resource Requirements *(Continued)*

Resource	Value	Usage Notes
JCNR vRouter Agent cores	.5	

Miscellaneous Requirements

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

Table 24: Miscellaneous Requirements

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

Set IOMMU. For example:
`echo Y > /sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts`
`echo Y > /sys/module/vfio/parameters/enable_unsafe_noiommu_mode`

Configure persistence for vfio and vfio-pci kernel modules after node reboot:
Add the module names to `/etc/modules-load.d/vfio.conf`:
`# sudo su`
`# cat /etc/modules-load.d/vfio.conf`
`vfio`
`vfio-pci`

Table 24: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

Configure IPv4 and IPv6 addresses for the interfaces allocated to JCNR. For example:
 	
`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`

Additional kernel modules need to be loaded on the host before deploying JCNR in L3 mode. These modules are usually available in `linux-modules-extra` or `kernel-modules-extra` packages. Run the following commands to add the kernel modules:
 	
`cat /etc/modules-load.d/crpd.conf`
`tun`
`fou`
`fou6`
`ipip`
`ip_tunnel`
`ip6_tunnel`
`mpls_gso`
`mpls_router`
`mpls_iptunnel`
`vrf`
`vxlan`
 	
NOTE: Applicable for L3 deployments only.

Run the `ip fou add port 6635 ipproto 137` command on the Linux host to enable kernel based forwarding.

Table 24: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

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

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 25: Cloud-Native Router Listening Ports

Protocol	Port	Description
TCP	8085	vRouter introspect–Used to gain internal statistical information about vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from JCNR control plane
TCP	9091	vRouter health check–cloud-native router checks to ensure **contrail-vrouter-dpdk** process is running, etc.
TCP	50052	gRPC port–JCNR listens on both IPv4 and IPv6
TCP	8081	JCNR Deployer Port
TCP	22	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	50051	**jsd** on cRPD
TCP	51051	**jsd** on cRPD

Table 25: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
UDP	50055	Syslog-NG

Download Options

To deploy JCNR on a Wind River deployment you can download the helm charts from the Juniper Support Site.

> **NOTE**:
> `https://enterprise.hub.juniper.net`

JCNR Licensing

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

> **NOTE**: Starting with JCNR Release 23.2, the JCNR license format has changed. Request a new license key from the JAL portal before deploying or upgrading to 23.2 or newer releases.

Customize JCNR 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 Attributes and Descriptions

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.

Table 26: Helm Chart Attributes and Descriptions

Key	Additional Key Configuration	Description
registry		Defines the docker registry for the vRouter, cRPD and jcnr-cni 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 vRouter, cRPD and jcnr-cni container images. This is a global key and takes precedence over "repository" paths under "common" section. The default value is `jcnr-container-prod/`.
imagePullSecret		(Optional) Defines the registry authentication credentials. You can configure credentials to either the Juniper repository or your private registry.

Table 26: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
	registryCredentials	Base64 representation of your Docker registry credentials. View the "Configure Repository Credentials" on page 301 topic for more information.
	secretName	Name of the secret object that will be created.
common		Defines repsitory paths and tags for the vRouter, cRPD and jcnr-cni container images. Use default unless using a private registry.
	repository	Defines the repository path. The default value is `atom-docker/cn2/bazel-build/dev/`. The global repository key takes precedence if defined.
	tag	Defines the image tag. The default value is configured to the appropriate tag number for the JCNR release version.
replicas		(Optional) Indicates the number of replicas for cRPD. If the value is not specified, then the default value 1 is considered. The value for this key must be specified for multi-node clusters. The value must be equal to the number of nodes to which JCNR must be deployed.
storageClass		Not applicable for non-cloud deployments.
awsregion		Not applicable for non-EKS deployments.
noLocalSwitching		(Optional) Prevents interfaces in a bridge domain from transmitting and receiving ethernet frame copies. Enter one or more comma separated VLAN IDs to ensure that the interfaces belonging to the VLAN IDs do not transmit frames to one another. This key is specific for L2 and L2-L3 deployments. Enabling this key provides the functionality on all access interfaces. For enabling the functionality on trunk interfaces, configure the no-local-switching key in the fabricInterface key.

Table 26: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description

fabricInterface		Provide a list of interfaces to be bound to the DPDK. You can also provide subnets instead of interface names. If both the interface name and the subnet are specified, then the interface name takes precedence over subnet/gateway combination. The subnet/gateway combination is useful when the interface names vary in a multi-node cluster.
		
		For example:
		
		`# L3 only`
		`- eth1:`
		`ddp: "off"`

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

	ddp	(Optional) Indicates the interface-level Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at NIC for traffic like GTPU, SCTP, etc. Setting options include auto, on, or off. The default setting is off. **NOTE**: The subnet/interface level `ddp` takes precedence over the global `ddp` configuration.
	interface_mode	Not applicable for Wind River deployment.
	vlan-id-list	Not applicable for Wind River deployment.

Table 26: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
	storm-control-profile	Not applicable for Wind River deployment.
	native-vlan-id	Not applicable for Wind River deployment.
	no-local-switching	Not applicable for Wind River deployment.
fabricWorkloadInterface		Not applicable for Wind River deployment.
log_level		Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave the log_level set to the default INFO unless instructed to change it by Juniper support.
log_path		The defined directory stores various JCNR related descriptive logs such as contrail-vrouter-agent.log, contrail-vrouter-dpdk.log, etc. The default value is /var/log/jcnr/.
syslog_notifications		Indicates the absolute path to the file that stores syslog-ng generated notifications in JSON format. The default value is /var/log/jcnr/jcnr_notifications.json.
corePattern		Indicates the core pattern to denote how the core file is generated. If this configuration is left blank, then JCNR pods will not overwrite the default pattern. **NOTE**: Set the corePattern value on host before deploying JCNR. You may change the value in `/etc/sysctl.conf`. For example, `kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`
coreFilePath		Indicates the path for the core file. If the value is left blank, then vRouter considers /var/crashes as the default value.

Table 26: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description

nodeAffinity		(Optional) Defines labels on nodes to determine where to place the vRouter pods.
		
		By default the vRouter pods are deployed to all worker 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 JCNR must 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		Set the value to /var/opt/cni/bin.
grpcTelemetryPort		(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50051.
grpcVrouterPort		(Optional) Enter a value for this parameter to override vRouter gRPC server default port of 50052.
vRouterDeployerPort		(Optional) Default value is 8081. Configure to override if the default port is unavailable.
restoreInterfaces		Set the value of this key to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts.

Table 26: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
bondInterfaceConfigs		Not applicable for Wind River deployment.
mtu		Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default value is 9000.
cpu_core_mask		Indicates the vRouter forward core mask. If qos is enabled, you will need to allocate 4 CPU cores (primary and siblings).
stormControlProfiles		Configure the rate limit profiles for BUM traffic on fabric interfaces in bytes per second.
dpdkCommandAdditionalArgs		Pass any additional dpdk cmd line parameters. The --yield_option 0 is set by default and it implies the dpdk forwarding cores will not yield the cpu cores it is assigned to. Additional 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"`
ddp		(Optional) Indicates the global Dynamic Device Personalization (DDP) configuration. DDP provides datapath optimization at NIC for traffic like GTPU, SCTP, etc. For a bond interface, all slave interface NICs must support DDP for the DDP configuration to be enabled. Setting options include auto, on, or off. The default setting is off. **NOTE**: The interface level `ddp` takes precedence over the global `ddp` configuration.
qosEnable		Set to false for Wind River Deployment.
vrouter_dpdk_uio_driver		The uio driver is `vfio-pci`.

Table 26: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
agentModeType		Can be dpdk or xdp. Setting agentModeType to dpdk will bringup dpdk datapath. Setting agentModeType to xdp uses ebpf. The default value is 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.
persistConfig		Set this flag to true if you wish jcnr-cni generated pod configuration to persist even after uninstallation. The option must be set only for L2 mode. The default value is false.

Sample Helm Charts

Helm Chart for L3 Only Deployment on Wind River Deployment

A working L3 only helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred
```

```
  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

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

  # storageClass: Name of the storage class for cRPD. This option is must for
  # cloud deployments such as AWS where gp2 can be used
  #storageClass: gp2
  # Set AWS Region for AWS deployments
  #awsregion: us-east-1
  #noLocalSwitching: [700]

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

```
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [700]
  #    storm-control-profile: rate_limit_pf1
  #    native-vlan-id: 100
  #    no-local-switching: true
  #- bond0:
  #    ddp: "auto" # auto/on/off  # ddp parameter is optional; options include auto or on or
off; default: off
  #    interface_mode: trunk
  #    vlan-id-list: [100, 200, 300, 700-705]
  #    storm-control-profile: rate_limit_pf1
  #    #native-vlan-id: 100
  #    #no-local-switching: true

  ##########################
  # L3  only
  - ens2f2:
      ddp: "auto"
  - ens1f1:
      ddp: "auto"
  ########################

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

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
```

```
  #- subnet: 10.40.1.0/24
  #  gateway: 10.40.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off
  #- subnet: 192.168.1.0/24
  #  gateway: 192.168.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off

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

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

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

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

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

  # 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/opt/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
  #vRouterDeployerPort: 8082


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

 # Enable bond interface configurations L2 only or L2 L3 deployment

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

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

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

  # uio driver will be vfio-pci or uio_pci_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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Customize JCNR Configuration

SUMMARY

Read this topic to understand how to customize JCNR configuration using a ConfigMap.

IN THIS SECTION

JCNR ConfigMap

Starting with Juniper Cloud-Native Router (JCNR) Release 23.3, JCNR supports customizing configuration using a ConfigMap when deployed in L3 mode. In cloud-based deployments, in the event of a node failure, the JCNR pods may be spawned on newer or different nodes. A ConfigMap decouples the configuration parameters from node names and is based on node labels instead. This enables the JCNR CNI deployer to consume the configuration parameters, apply them to the cRPD configuration template and render the configuration, as long as a matching label is available for the node.

A ConfigMap is an API object to store data in key-values pairs. A ConfigMap defines per node variables that are consumed by nodes matching the label. The key-value pairs are used to render the configuration via a go template. The configured template must be available in the `Juniper_Cloud_Native_Router_`*`release_number`*`/helmchart/charts/jcnr-cni/files/` directory for the configuration to be applied to the cRPD pods.

NOTE: You must apply the ConfigMap before installing JCNR to create cRPD pods with custom configuration. The cRPD pod must be deleted and respawned should you wish to apply the configuration parameters any time after JCNR installation. The configuration parameters are applied by default to any newly spawned cRPD pods. The JCNR customization via ConfigMap is optional.

NOTE: JCNR also supports customization via node annotations for backward compatibility with previous releases. Considering that node annotations are coupled with node names, it is highly recommended to customize JCNR via ConfigMaps, specifically for cloud deployments. Refer to Customize JCNR Configuration using node annotations for more information.

Configuration Example

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

You define the key-value pair for different node labels in your cluster. An example of the `jcnr-params-configmap.yaml` file is provided below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: jcnr-params
  namespace: jcnr
data:
  jcnr1: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0001.00",
      "IPv4LoopbackAddr": "110.1.1.2",
      "srIPv4NodeIndex": "2000",
      "srIPv6NodeIndex": "3000",
      "BGPIPv4Neighbor": "110.1.1.254",
      "BGPLocalAsn": "64512"
    }
  jcnr2: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0000.00",
      "IPv4LoopbackAddr": "110.1.1.3",
      "srIPv4NodeIndex": "2001",
      "srIPv6NodeIndex": "3001",
      "BGPIPv4Neighbor": "110.1.2.254",
      "BGPLocalAsn": "64512"
    }
```

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

```
apply-groups [custom];
groups {
    custom {
        interfaces {
            lo0 {
```

```
                unit 0 {
                {{if .Params.isoLoopbackAddr}}
                    family iso {
                        address {{.Params.isoLoopbackAddr}};
                    }
                {{end}}
                    family inet {
                        address {{.Params.IPv4LoopbackAddr}};
                    }
                }
            }
        }
        routing-options {
            router-id {{.Params.IPv4LoopbackAddr}}
            route-distinguisher-id {{.Params.IPv4LoopbackAddr}}
        }
        protocols {
            isis {
                interface all;
                {{if and .Env.SRGB_START_LABEL .Env.SRGB_INDEX_RANGE}}
                source-packet-routing {
                    srgb start-label {{.Env.SRGB_START_LABEL}} index-range
{{.Env.SRGB_INDEX_RANGE}};
                    node-segment {
                        {{if .Params.srIPv4NodeIndex}}
                        ipv4-index {{.Params.srIPv4NodeIndex}};
                        {{end}}
                        {{if .Params.srIPv6NodeIndex}}
                        ipv6-index {{.Params.srIPv6NodeIndex}};
                        {{end}}
                    }
                }
                {{end}}
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
        policy-options {
```

```
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then community add udp;
            }
            community udp members encapsulation:0L:13;
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address {{.Params.IPv4LoopbackAddr}};
                    local-as {{.Params.BGPLocalAsn}};
                    neighbor {{.Params.BGPIPv4Neighbor}};
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                }
            }
        }
        routing-options {
                dynamic-tunnels {
                dyn-tunnels {
                    source-address {{.Params.IPv4LoopbackAddr}};
                    udp;
                    destination-networks {{.Params.BGPIPv4Neighbor}}/32;
                }
            }
        }
    }
}
```

NOTE: You can define additional cRPD configuration hierarchies in the template. The values to be rendered from the ConfigMap defined in the `jcnr-params-configmap.yaml` must be defined as {{.Params.*var-name*}}. Any environment variables, such as variables defined in `values.yaml`, must be defined as {{.Env.*variable_name*}}.

Complete the following steps to apply the customizations.

1. Label each node based on the keys used in the ConfigMap.

```
kubectl label nodes <node_name1> jcnr.juniper.net/params-profile=jcnr1
kubectl label nodes <node_name2> jcnr.juniper.net/params-profile=jcnr2
```

2. Apply the ConfigMap to the cluster nodes using the command provided below:

```
# kubectl apply -f jcnr-params-configmap.yaml
configmap/jcnr-params created
```

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

```
# cp Juniper_Cloud_Native_Router_release_number/helmchart/cRPD_examples/jcnr-cni-custom-
config-cm.tmpl Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/
#
```

4. Deploy the cloud-native router components, including the cRPD. Once the installation completes, *access the cRPD CLI* and issue the `show configuration | display set` command in the `cli` mode to view the custom configuration you applied.

```
root@jcnr-01> show configuration
## Last commit: 2023-06-23 08:30:42 EDT by root
version 20230608.143922_builder.r1342735;
groups {
    base { /* OMITTED */ };
    custom {
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 110.1.1.2/32;
                    }
                    family iso {
                        address 49.0004.1000.0000.0001.00;
                    }
                }
            }
        }
```

```
policy-options {
    # policy to signal dynamic UDP tunnel attributes to BGP routes
    policy-statement udp-export {
        then {
            community add udp;
        }
    }
    community udp members encapsulation:0L:13;
}
routing-options {
    route-distinguisher-id 110.1.1.2;
    router-id 110.1.1.2;
    dynamic-tunnels {
        dyn-tunnels {
            source-address 110.1.1.2;
            udp;
            destination-networks {
                110.1.1.254/32;
            }
        }
    }
}
protocols {
    bgp {
        group jcnrbgp1 {
            type internal;
            local-address 110.1.1.2;
            family inet-vpn {
                unicast;
            }
            family inet6-vpn {
                unicast;
            }
            local-as 64512;
            neighbor 110.1.1.254;
        }
    }
    isis {
        interface all;
        source-packet-routing {
            srgb start-label 400000 index-range 4000;
            node-segment {
                ipv4-index 2000;
```

```
                        ipv6-index 3000;
                    }
                }
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
    }
    cni { /* OMITTED */ };
    internal { /* OMITTED */ };
}
apply-groups [ custom base internal ];
```

Modifying the ConfigMap

If you wish to change the ConfigMap any time after JCNR installation, you must delete the cRPD pod and respawn it using the following command:

```
kubectl patch sts kube-crpd-worker-sts -n jcnr -p '{"spec":{"template":{"metadata":
{"annotations":{"configmap-reload/timestamp":"'$(date +%s)'"}}}}}'
```

This triggers a rolling restart of all cRPD pods. Alternatively, you can identify the cRPD pods on nodes for which the ConfigMap has changed and manually delete the pod. The ConfigMap changes will be applied automatically to any respawned pods.

Troubleshooting

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

CHAPTER 7

Install Cloud-Native Router on Microsoft Azure Cloud Platform

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

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

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 254 section to ensure the setup has all the required configuration.
2. Download the JCNR helm charts, **Juniper_Cloud_Native_Router_*release-number*.tgz**, to the directory of your choice. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
3. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

```
tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
```

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

```
cd Juniper_Cloud_Native_Router_release-number
```

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

5. View the contents in the current directory.

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

6. Enter the root password for your host server and your Juniper Cloud-Native Router license file into the **secrets/jcnr-secrets.yaml** file. You must enter the password and license in base64 encoded format.

   You can view the sample contents of the **jcnr-secrets.yaml** file below:

```
---
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>
```

   To encode the password, create a file with the plain text password on a single line. Then issue the command:

```
base64 -w 0 rootPasswordFile
```

To encode the license key, copy the license key into a file on your host server and issue the command:

```
base64 -w 0 licenseFile
```

You must copy the base64 outputs and paste them into the **secrets/jcnr-secrets.yaml** file in the appropriate locations.

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

Apply the **secrets/jcnr-secrets.yaml** to the Kubernetes system.

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

> **NOTE**: Starting with JCNR Release 23.2, the JCNR license format has changed. Request a new license key from the JAL portal before deploying or upgrading to 23.2 or newer releases.

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

   See, "Customize JCNR Helm Chart for Azure Deployment" on page 263 for descriptions of the helm chart configurations and a sample helm chart for Azure deployment..

8. Optionally, customize JCNR configuration.

   See, "Customize JCNR Configuration " on page 54 for creating and applying the cRPD customizations.

9. Label the nodes to which JCNR must be installed based on the `nodeaffinity` defined in the `values.yaml`. For example:

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

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

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

```
helm install jcnr .
```

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

11. 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-23.3.0                23.3.0
```

Verify Installation

This section enables you to confirm a successful JCNR deployment.

1. Verify the state of the JCNR pods by issuing the `kubectl get pods -A` command.

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

```
kubectl get pods -A
```

```
NAMESPACE         NAME                                    READY   STATUS
RESTARTS          AGE
contrail-deploy   contrail-k8s-deployer-579cd5bc74-g27gs   1/1     Running
0                 103s
contrail          contrail-vrouter-masters-lqjqk          3/3     Running
0                 87s
jcnr              kube-crpd-worker-sts-0                  1/1     Running
0                 103s
jcnr              syslog-ng-ds5qd                         1/1     Running
0                 103s
kube-system       calico-kube-controllers-5f4fd8666-m78hk 1/1     Running
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 JCNR daemonsets by issuing the `kubectl get ds -A` command.

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

```
kubectl get ds -A
```

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

3. Verify the JCNR statefulsets by issuing the `kubectl get statefulsets -A` command.

   The command output provides the statefulsets.

```
kubectl get statefulsets -A
```

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

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

   a. View the *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 JCNR 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 JCNR 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 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 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
            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

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

Table 27: Cloud-Native Router Minimum Host System Requirements

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	The tested kernel version is 4.18.0-477.15.1.el8_8.cloud.x86_64

Table 27: Cloud-Native Router Minimum Host System Requirements *(Continued)*

Component	Value/Version	Notes
Kubernetes (K8s)	Version 1.25.x	The tested K8s version is 1.25.5
Calico	Version 3.25.1	
Multus	Version 4.0	
Helm	3.9.x	
Container-RT	containerd	

Resource Requirements

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

Table 28: Cloud-Native Router Resource Requirements

Resource	Value	Usage Notes
Data plane forwarding cores	2 cores (2P + 2S)	
Service/Control Cores	0	
UIO Driver	uio_hv_generic	To enable, follow the below steps cat /etc/modules-load.d/k8s.conf uio uio_hv_generic ib_uverbs mlx4_ib The above libraries are provided by ibverbs package.

Table 28: Cloud-Native Router Resource Requirements *(Continued)*

Resource	Value	Usage Notes

Hugepages (1G)	6 Gi	Add `GRUB_CMDLINE_LINUX_DEFAULT` values in **/etc/default/grub** and reboot the host. For example:
		
		`GRUB_CMDLINE_LINUX_DEFAULT="consol e=tty1 console=ttyS0 default_hugepagesz=1G hugepagesz=1G hugepages=6 intel_iommu=on iommu=pt"`
		
		Update grub and reboot the host. For example:
		
		`grub2-mkconfig -o /boot/grub2/ grub.cfg`
		
		Verify the hugepage is set by executing the following commands:
		
		`cat /proc/cmdline`
		`grep -i hugepages /proc/meminfo`

JCNR Controller cores	.5	
JCNR vRouter Agent cores	.5	

Miscellaneous Requirements

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

Table 29: Miscellaneous Requirements

Cloud-Native Router Release Miscellaneous Requirements

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

Additional kernel modules need to be loaded on the host before deploying JCNR in L3 mode. These modules are usually available in `linux-modules-extra` or `kernel-modules-extra` packages. Run the following commands to add the kernel modules:
 	
`cat /etc/modules-load.d/crpd.conf`
`tun`
`fou`
`fou6`
`ipip`
`ip_tunnel`
`ip6_tunnel`
`mpls_gso`
`mpls_router`
`mpls_iptunnel`
`vrf`
`vxlan`
 	
NOTE: Applicable for L3 deployments only.

Run the `ip fou add port 6635 ipproto 137` command on the Linux host to enable kernel based forwarding.

Table 29: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements
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 JCNR 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, 2.51.2.0/23, 2.51.1.0/24, 2.2.2.2/32, 3.3.3.3/32 7. `Protocols`: all 8. `Enforcement`: Enabled where 10.2.0.0/24 is the subnet to which JCNR is attached and 2.51.2.0/24, 2.51.1.0/24, 2.2.2.2/32, 3.3.3.3/32 are loopback IP ranges.
JCNR for Azure supports IPv4 only.

Table 29: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

Ensure accelerated networking is enabled for the fabric interface. If accelerated networking is enabled properly, two interfaces become available for the fabric interface. For example:
 	
`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`
 	
When configuring the fabric interface in the helm chart, you must provide the interface with `hv_netvsc` binded to it. Issue the `ethtool -i` *`interface_name`* command to verify it. For example:
 	
`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`
 	
NOTE: Do not enable accelerated networking for the management interface.

Table 29: Miscellaneous Requirements *(Continued)*

Cloud-Native Router Release Miscellaneous Requirements

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

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 30: Cloud-Native Router Listening Ports

Protocol	Port	Description
TCP	8085	vRouter introspect–Used to gain internal statistical information about vRouter
TCP	8072	Telemetry Information-Used to see telemetry data from JCNR control plane
TCP	9091	vRouter health check–cloud-native router checks to ensure **contrail-vrouter-dpdk** process is running, etc.
TCP	50052	gRPC port–JCNR listens on both IPv4 and IPv6
TCP	8081	JCNR Deployer Port
TCP	22	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	50051	**jsd** on cRPD
TCP	51051	**jsd** on cRPD

Table 30: Cloud-Native Router Listening Ports *(Continued)*

Protocol	Port	Description
UDP	50055	Syslog-NG

Download Options

To deploy JCNR on Azure you can download the helm charts from the Juniper Support Site.

NOTE:
`https://enterprise.hub.juniper.net`

JCNR Licensing

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

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

Customize JCNR 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 Attributes and Descriptions

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

Table 31: Helm Chart Attributes and Descriptions

Key	Additional Key Configuration	Description
registry		Defines the docker registry where the vRouter, cRPD and jcnr-cni container images are hosted. The default value is `enterprise-hub.juniper.net`.
repository		(Optional) Defines the repository path for the vRouter, cRPD and jcnr-cni container images. This is a global key and takes precedence over "repository" paths under "common" section. The default value is `jcnr-container-prod/`.
imagePullSecret		(Optional) Defines the registry authentication credentials. You can configure credentials to either the Juniper repository or your private registry.

Table 31: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
	registryCredentials	Base64 representation of your Docker registry credentials. View the "Configure Repository Credentials" on page 301 topic 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.
	repository	Defines the repository path. The default value is `atom-docker/cn2/bazel-build/dev/`. The global repository key takes precedence if defined.
	tag	Defines the image tag. The default value is configured to the appropriate tag number for the JCNR release version.
replicas		(Optional) Indicates the number of replicas for cRPD. If the value is not specified, then the default value 1 is considered. The value for this key must be specified for multi-node clusters and must match the number of nodes to which JCNR must be deployed.
storageClass		Not applicable for Azure deployments.
awsregion		Not applicable for Azure deployments.
noLocalSwitching		Not applicable for Azure deployments.

Table 31: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	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.
		For example:
		`# L3 only`
		`- eth1:`
		`ddp: "off"`
		`- eth2:`
		`ddp: "off"`

	subnet	Not applicable for Azure deployments.
	ddp	Not applicable for Azure deployments.
	interface_mode	Not applicable for Azure deployments.
	vlan-id-list	Not applicable for Azure deployments.
	storm-control-profile	Not applicable for Azure deployments.
	native-vlan-id	Not applicable for Azure deployments.
	no-local-switching	Not applicable for Azure deployments.
fabricWorkloadInterface		Not applicable for Azure deployments.
log_level		Defines the log severity. Available value options are: DEBUG, INFO, WARN, and ERR. **NOTE**: Leave the log_level set to INFO unless instructed to change it by Juniper support.

Table 31: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
log_path		The defined directory stores various JCNR related descriptive logs such as contrail-vrouter-agent.log, contrail-vrouter-dpdk.log, etc.
syslog_notifications		Indicates the absolute path to the file that stores syslog-ng generated notifications in JSON format.
corePattern		Indicates the core pattern to denote how the core file is generated. If this configuration is left blank, then JCNR pods will not overwrite the default pattern. **NOTE**: Set the corePattern value on host before deploying JCNR. You may change the value in `/etc/sysctl.conf`. For example, `kernel.core_pattern=/var/crash/core_%e_%p_%i_%s_%h_%t.gz`
coreFilePath		Indicates the path for the core file. If the value is left blank, then vRouter considers /var/crashes as the default value.

nodeAffinity		(Optional) Defines labels on nodes to determine where to place the vRouter pods. By default the vRouter pods are deployed to all worker 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 JCNR must 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 Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	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 cni path with a path of used by your distribution e.g. /var/opt/cni/bin.
grpcTelemetryPort		(Optional) Enter a value for this parameter to override cRPD telemetry gRPC server default port of 50051.
grpcVrouterPort		(Optional) Enter a value for this parameter to override vRouter gRPC server default port of 50052.
restoreInterfaces		Set the value of this key to true to restore the interfaces back to their original state in case the vRouter pod crashes or restarts.
vRouterDeployerPort		(Optional) Default value is 8081. Configure to override if the default port is unavailable.
bondInterfaceConfigs		Not applicable for Azure deployments.
mtu		Maximum Transmission Unit (MTU) value for all physical interfaces (VFs and PFs). Default value is 9000.
cpu_core_mask		Indicates the vRouter forward core mask. If qos is enabled, you will need to allocate 4 CPU cores (primary and siblings).
stormControlProfiles		Not applicable for Azure deployments.

Table 31: Helm Chart Attributes and Descriptions *(Continued)*

Key	Additional Key Configuration	Description
dpdkCommandAdditionalArgs		Pass any additional dpdk cmd line parameters. The --yield_option 0 is set by default and it implies the dpdk forwarding cores will not yield the cpu cores it is assigned to. Additional 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"`
ddp		Not applicable for Azure deployments.
qosEnable		Set to false for Azure deployments.
vrouter_dpdk_uio_driver		The uio driver is `uio_hv_generic`.
agentModeType		Can be dpdk or xdp. Setting agentModeType to dpdk will bringup dpdk datapath. Setting agentModeType to xdp uses ebpf. The default value is 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.
persistConfig		Set this flag to true if you wish jcnr-cni generated pod configuration to persist even after uninstallation. The option must be set only for L2 mode. The default value is false.

Helm Chart for Azure Deployment

A working L3 only helm chart sample is shown below. The configured sections are highlighted in **bold**:

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

  # uncomment below if you are using a private registry that needs authentication
  # registryCredentials - Base64 representation of your Docker registry credentials
  # secretName - Name of the Secret object that will be created
  #imagePullSecret:
    #registryCredentials: <base64-encoded-credential>
    #secretName: regcred

  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85
    crpd:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/amd64/
      tag: 23.4-20231215-50817e3
    telemetryExporter:
      repository: atom-docker/cn2/bazel-build/dev/x86_64/
      tag: R23.4-85

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

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

```
  # Set AWS Region for AWS deployments
  #awsregion: us-east-1
  #noLocalSwitching: [700]

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

  ########################
  # L3  only
  - ens2f2:
      ddp: "auto"
  - ens1f1:
      ddp: "auto"
  ########################
```

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

  # Provide subnets instead of interface names
  # Interfaces will be auto-detected in each subnet
  # Only one of the interfaces or subnet range must
  # be configured. This form of input is particularly
  # helpful when the interface names vary in a multi-node
  # K8s cluster
  #- subnet: 10.40.1.0/24
  #  gateway: 10.40.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off
  #- subnet: 192.168.1.0/24
  #  gateway: 192.168.1.1
  #  ddp: "off"                    # ddp parameter is optional; options include auto or on or
off; default: off

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

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

```
  log_level: "INFO"

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

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

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

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

  # cni_bin_dir: Path where the CNI binary will be put; default: /opt/cni/bin
  # this may be overriden in distributions other than vanilla
K8s
  # e.g. OpenShift - you may use /var/lib/cni/bin or /etc/kubernetes/cni/net.d
  #cni_bin_dir:  /var/lib/cni/bin

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

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

  # vRouterDeployerPort: use this parameter to override vRouter deployer port default port of
8081
```

```
  #vRouterDeployerPort: 8082


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

 # Enable bond interface configurations L2 only or L2 L3 deployment

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

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

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

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

  dpdkCommandAdditionalArgs: "--yield_option 0"

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

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

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

  # 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

#jcnr-cni:
  # persistConfig: set this flag to true if you wish jcnr-cni generated pod configuration to
persist even after uninstallation
  # use this option only in case of l2 mode
  # default value is false if not specfied
  #persistConfig: true
```

Customize JCNR Configuration

SUMMARY

Read this topic to understand how to customize JCNR configuration using a ConfigMap.

IN THIS SECTION

JCNR ConfigMap

Starting with Juniper Cloud-Native Router (JCNR) Release 23.3, JCNR supports customizing configuration using a ConfigMap when deployed in L3 mode. In cloud-based deployments, in the event of a node failure, the JCNR pods may be spawned on newer or different nodes. A ConfigMap decouples the configuration parameters from node names and is based on node labels instead. This enables the

JCNR CNI deployer to consume the configuration parameters, apply them to the cRPD configuration template and render the configuration, as long as a matching label is available for the node.

A ConfigMap is an API object to store data in key-values pairs. A ConfigMap defines per node variables that are consumed by nodes matching the label. The key-value pairs are used to render the configuration via a go template. The configured template must be available in the `Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/` directory for the configuration to be applied to the cRPD pods.

NOTE: You must apply the ConfigMap before installing JCNR to create cRPD pods with custom configuration. The cRPD pod must be deleted and respawned should you wish to apply the configuration parameters any time after JCNR installation. The configuration parameters are applied by default to any newly spawned cRPD pods. The JCNR customization via ConfigMap is optional.

NOTE: JCNR also supports customization via node annotations for backward compatibility with previous releases. Considering that node annotations are coupled with node names, it is highly recommended to customize JCNR via ConfigMaps, specifically for cloud deployments. Refer to Customize JCNR Configuration using node annotations for more information.

Configuration Example

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

You define the key-value pair for different node labels in your cluster. An example of the `jcnr-params-configmap.yaml` file is provided below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: jcnr-params
  namespace: jcnr
data:
  jcnr1: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0001.00",
      "IPv4LoopbackAddr": "110.1.1.2",
```

```
      "srIPv4NodeIndex": "2000",
      "srIPv6NodeIndex": "3000",
      "BGPIPv4Neighbor": "110.1.1.254",
      "BGPLocalAsn": "64512"
    }
  jcnr2: |
    {
      "isoLoopbackAddr": "49.0004.1000.0000.0000.00",
      "IPv4LoopbackAddr": "110.1.1.3",
      "srIPv4NodeIndex": "2001",
      "srIPv6NodeIndex": "3001",
      "BGPIPv4Neighbor": "110.1.2.254",
      "BGPLocalAsn": "64512"
    }
```

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

```
apply-groups [custom];
groups {
    custom {
        interfaces {
            lo0 {
                unit 0 {
                {{if .Params.isoLoopbackAddr}}
                    family iso {
                        address {{.Params.isoLoopbackAddr}};
                    }
                {{end}}
                    family inet {
                        address {{.Params.IPv4LoopbackAddr}};
                    }
                }
            }
        }
        routing-options {
            router-id {{.Params.IPv4LoopbackAddr}}
            route-distinguisher-id {{.Params.IPv4LoopbackAddr}}
        }
        protocols {
            isis {
                interface all;
```

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

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

NOTE: You can define additional cRPD configuration hierarchies in the template. The values to be rendered from the ConfigMap defined in the `jcnr-params-configmap.yaml` must be defined as {{.Params.*var-name*}}. Any environment variables, such as variables defined in `values.yaml`, must be defined as {{.Env.*variable_name*}}.

Complete the following steps to apply the customizations.

1. Label each node based on the keys used in the ConfigMap.

```
kubectl label nodes <node_name1> jcnr.juniper.net/params-profile=jcnr1
kubectl label nodes <node_name2> jcnr.juniper.net/params-profile=jcnr2
```

2. Apply the ConfigMap to the cluster nodes using the command provided below:

```
# kubectl apply -f jcnr-params-configmap.yaml
configmap/jcnr-params created
```

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

```
# cp Juniper_Cloud_Native_Router_release_number/helmchart/cRPD_examples/jcnr-cni-custom-
config-cm.tmpl Juniper_Cloud_Native_Router_release_number/helmchart/charts/jcnr-cni/files/
#
```

4. Deploy the cloud-native router components, including the cRPD. Once the installation completes, *access the cRPD CLI* and issue the `show configuration | display set` command in the `cli` mode to view the custom configuration you applied.

```
root@jcnr-01> show configuration
## Last commit: 2023-06-23 08:30:42 EDT by root
version 20230608.143922_builder.r1342735;
groups {
    base { /* OMITTED */ };
    custom {
        interfaces {
            lo0 {
                unit 0 {
                    family inet {
                        address 110.1.1.2/32;
                    }
                    family iso {
                        address 49.0004.1000.0000.0001.00;
                    }
                }
            }
        }
        policy-options {
            # policy to signal dynamic UDP tunnel attributes to BGP routes
            policy-statement udp-export {
                then {
                    community add udp;
                }
            }
            community udp members encapsulation:0L:13;
        }
        routing-options {
            route-distinguisher-id 110.1.1.2;
            router-id 110.1.1.2;
            dynamic-tunnels {
                dyn-tunnels {
                    source-address 110.1.1.2;
                    udp;
                    destination-networks {
                        110.1.1.254/32;
                    }
                }
```

```
            }
        }
        protocols {
            bgp {
                group jcnrbgp1 {
                    type internal;
                    local-address 110.1.1.2;
                    family inet-vpn {
                        unicast;
                    }
                    family inet6-vpn {
                        unicast;
                    }
                    local-as 64512;
                    neighbor 110.1.1.254;
                }
            }
            isis {
                interface all;
                source-packet-routing {
                    srgb start-label 400000 index-range 4000;
                    node-segment {
                        ipv4-index 2000;
                        ipv6-index 3000;
                    }
                }
                level 1 disable;
            }
            ldp {
                interface all;
            }
            mpls {
                interface all;
            }
        }
    }
    cni { /* OMITTED */ };
    internal { /* OMITTED */ };
}
apply-groups [ custom base internal ];
```

Modifying the ConfigMap

If you wish to change the ConfigMap any time after JCNR installation, you must delete the cRPD pod and respawn it using the following command:

```
kubectl patch sts kube-crpd-worker-sts -n jcnr -p '{"spec":{"template":{"metadata":
{"annotations":{"configmap-reload/timestamp":"'$(date +%s)'"}}}}}'
```

This triggers a rolling restart of all cRPD pods. Alternatively, you can identify the cRPD pods on nodes for which the ConfigMap has changed and manually delete the pod. The ConfigMap changes will be applied automatically to any respawned pods.

Troubleshooting

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

8
CHAPTER

Deploying Service Chain (cSRX) with JCNR

Deploying Service Chain (cSRX) with JCNR

IN THIS SECTION

Read this section to customize and deploy a security services instance (cSRX) with the Cloud-Native Router.

Starting Release 23.4, the Juniper Cloud-Native Router (JCNR) can be integrated with Juniper's containerized SRX (cSRX) platform to provide security services such as IPsec. This functionality is achieved 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.

Customize cSRX Helm Chart

The cSRX service chaining instance is deployed via a helm chart. The configuration parameters are provided via the `values.yaml` manifest file. 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` manifest 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` manifest file. Key configuration options include:

- 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 JCNR 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 JCNR to steer traffic from JCNR to cSRX and to steer IPsec traffic from the remote IPsec gateway to the cSRX to apply the security service chain.

Here is a sample values.yaml for 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:
    image: junos-csrx-init
    tag: 23.4R1.9
    imagePullPolicy: IfNotPresent
    resources:
      #limits:
      #  memory: 1Gi
      #  cpu: 1
      #requests:
      #  memory: 1Gi
      #  cpu: 1

  csrx:
    image: junos-csrx
    tag: 23.4R1.9
    imagePullPolicy: IfNotPresent
    resources:
      limits:
        hugepages-1Gi: 4Gi
        memory: 4Gi
      requests:
        hugepages-1Gi: 4Gi
        memory: 4Gi

# 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: 171.1.1.1/30          # should match ipSecTunnelConfigs localAddress if configured
    gateway: 171.1.1.2        # gateway configuration
    #ip6: 171:1:1::1/64        # optional
    #ip6Gateway: 171:1:1::2    # optional
    routes:                   # this field is optional
    - "181.1.1.0/24"
    #- "200.1.1.0/24"
  - 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
    - "222.1.1.0/24"
    #- "192.1.1.0/24"

ipSecTunnelConfigs:       # untrust
  - interface: eth1       ## section ike-phase1, proposal, policy, gateway
    gateway: 181.1.1.1
```

```
    localAddress: 171.1.1.1
    authenticationAlgorithm: sha-256
    encryptionAlgorithm: aes-256-cbc
    preSharedKey: "$9$zt3l3AuIRhev8FnNVsYoaApu0RcSyev8XO1NVYoDj.P5F9AyrKv8X"
    trafficSelector:
    - name: ts1
      localIP: 222.1.1.0/24 ## IP cannot be 0.0.0.0/0
      remoteIP: 111.1.1.0/24 ## IP cannot be 0.0.0.0/0

jcnr_config:
  - name: eth2
    routes:
      - "111.1.1.0/24"

csrx_ctrl_cpu: "0x01"

csrx_data_cpu: "0x0A"
```

Install cSRX

The cSRX service chain is deployed after the JCNR deployment. Read this section to install the cSRX instance.

1. Ensure you have the latest cSRX installation bundle downloaded and expanded. You must be in junos_csrx_*release_number* directory.
2. The cSRX container images are required for deployment. You may choose one of the following options:
   a. Download and deploy images from the Juniper repository—`enterprise-hub.juniper.net`. Review the Configure Repository Credentials topic for instructions on how to configure repository credentials in the deployment helm chart.
   b. You can upload the cSRX images either to a local docker or to your own docker respository using the `docker load` command. The images are available in the junos_csrx_*release_number*/images directory.

```
docker load -i images/csrx-images.tgz
```

3. Enter the cSRX license into the `secrets/csrx-secrets.yaml` file. You must enter the password and license in base64 encoded format. You can view the sample contents of the `csrx-secrets.yaml` file below:

```
apiVersion: v1
kind: Secret
metadata:
  name: service-chain-instance
  namespace: jcnr
data:
  csrx_license: |
    <add your license in base64 format>
```

To encode the license file, copy the license file onto your host server and issue the command:

```
base64 -w 0 licenseFile
```

You must copy the base64 output and paste it into the `secrets/csrx-secrets.yaml` file in the appropriate location.

NOTE: You must obtain your license file from your account team and install it in the 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.

4. Apply the `csrx-secrets.yaml` to the Kubernetes system.

```
kubectl apply -f secrets/secrets.yaml
secret/service-chain-instance created
```

5. Ensure all components of JCNR are up and running before installing the cSRX instance.

6. Ensure you have customized the cSRX helm chart. Navigate to the junos_csrx_*release_number*/ helmchart directory. Expand the bundle to view the helmcharts. Refer to the example given above to configure the values.yaml. Navigate to the junos_csrx_*release_number*/helmchart/junos-csrx directory and issue the following command to install the cSRX instance.

```
helm install csrx .
```

RELATED DOCUMENTATION

No Link Title

9

CHAPTER

Manage

Manage Juniper Cloud-Native Router

IN THIS SECTION

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

Upgrading JCNR

You can upgrade from JCNR release 23.2 to 23.3 using the following steps:

1. Download the tarball, **Juniper_Cloud_Native_Router_*release_number*.tgz**, to the directory of your choice. You must perform the file transfer in binary mode when transferring the file to your server, so that the compressed tar file expands properly.
2. Expand the file **Juniper_Cloud_Native_Router_*release-number*.tgz**.

   ```
   tar xzvf Juniper_Cloud_Native_Router_release-number.tgz
   ```

3. Change directory to Juniper_Cloud_Native_Router_*release-number*.

   ```
   cd Juniper_Cloud_Native_Router_release-number
   ```

   NOTE: **Juniper_Cloud_Native_Router_*release-number***

4. Customize the helm chart for your deployment using the `helmchart/values.yaml` file to match the helm chart configuration in your current installation.

5. Deploy the Juniper Cloud-Native Router using the helm chart. Navigate to the helmchart directory and run the following command:

```
helm upgrade jcnr .
Release "jcnr" has been upgraded. Happy Helming!
NAME: jcnr
LAST DEPLOYED: Thu Sep 21 03:58:28 2023
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
```

6. Confirm Juniper Cloud-Native Router deployment.

```
helm ls
```

```
NAME    NAMESPACE       REVISION        UPDATED
STATUS          CHART           APP VERSION
jcnr    default         2               2023-09-21 03:58:28.024244633 -0400 EDT
deployed        jcnr-23.3.0     23.3.0
```

Downgrading JCNR

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

Uninstalling JCNR

JCNR can be uninstalled by using the following command:

```
helm uninstall jcnr
```

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

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

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

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

10
CHAPTER

Troubleshoot

Troubleshoot Deployment Issues

SUMMARY

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

IN THIS SECTION

Troubleshoot Deployment Issues

IN THIS SECTION

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

Table 32: Investigate Deployment Issues

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

Table 32: Investigate Deployment Issues *(Continued)*

Potential issue	What to check	Related Commands

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

Table 32: Investigate Deployment Issues *(Continued)*

Potential issue	What to check	Related Commands

cRPD Pod in CrashLoopBackOff state	• Check if startup/liveness probe is failing or vrouter pod not running
	• rpd-vrouter-agent gRPC connection not UP
	• Composed configuration is invalid or config template is invalid
		• `kubectl get pods -A`
		`kubectl -n jcnr describe pod <crpd-pod-name>`
		`tail -f /var/log/jcnr/jcnr-cni.log`
		`tail -f /var/log/jcnr/jcnr_notifications.json`
		• See *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 pod for errors	`kubectl logs contrail-k8s-deployer-<pod-hash> -n contrail-deploy`

Verify Cloud-Native Router Controller Configuration

The cloud-native router deployment process creates a configuration file for the cloud-native router controller (cRPD) as a result of entries in the **values.yaml** file for L2 mode and custom configuration via node annotations in L3 mode. You can view this configuration file to see the details of the cRPD

configuration. To view the cRPD configuration, navigate to the `/var/run/jcnr` folder to access the configuration file details and view the contents of the configuration file.

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

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

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

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

In an AWS EKS deployment you can review the rendered custom configuration by *accessing the cRPD CLI* and reviewing the contents of the `/config` directory.

View Log Files

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

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

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

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

Uninstallation Issues

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

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

11

CHAPTER

Appendix

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).

Configure Repository Credentials

SUMMARY

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

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

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

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

   Enter your login credentials when prompted.

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

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

   Take a look at the encoded credentials.

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

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

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

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

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

   You should see the encoded credentials in the manifests.

Juniper Technology Previews (Tech Previews)

Tech Previews enable you to test functionality and provide feedback during the development process of innovations that are not final production features. The goal of a Tech Preview is for the feature to gain

wider exposure and potential full support in a future release. Customers are encouraged to provide feedback and functionality suggestions for a Technology Preview feature before it becomes fully supported.

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

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

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

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