JUNIPER NETWORKS | Engineering Simplicity

Juniper Cloud-Native Router Deployment Guide

Published
2022-10-03

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

1

CHAPTER

Overview

What Is the Juniper® Cloud-Native Router?

IN THIS SECTION

Overview

Juniper Cloud-Native Router (JCNR) is a container-based software solution that combines the JCNR-controller (cRPD-based control plane) and the JCNR-vRouter (DPDK-enabled forwarding/data plane). With the cloud-native router, you can enable Junos OS-based switching control with enhanced forwarding capabilities.

JCNR-controller running on a Kubernetes (K8s) compute-host provides control plane management functionality and uses the forwarding capabilities provided by either the Linux kernel or the DPDK-enabled JCNR-vRouter.

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, JCNR-Controller uses gRPC-based services to exchange messages and to communicate with JCNR-vRouter, thus creating the fully functional Cloud-Native Router. This close communication allows you to:

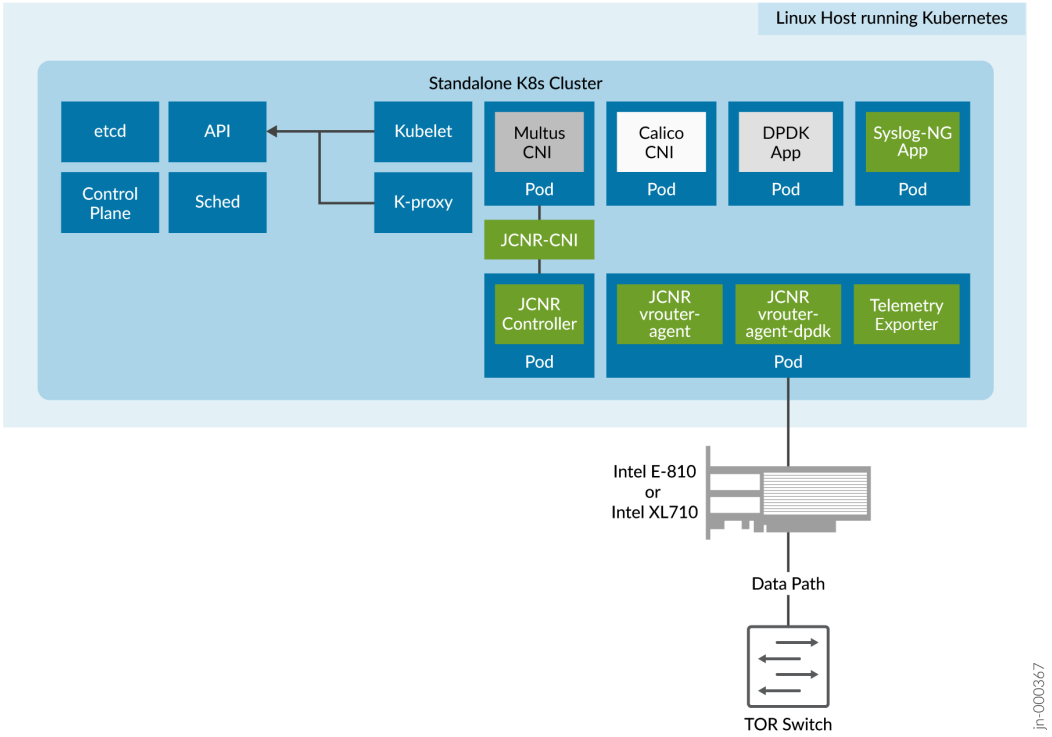
- Learn about fabric and workload interfaces
- Provision DPDK- or kernel-based interfaces for K8s pods as needed
- Configure IPv4 and IPv6 address allocation for Pods

Benefits

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

- Highly scalable solution

Figure 1: Components of Juniper Cloud-Native Router



Kubernetes

Kubernetes (K8s) 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 K8s pod consists of one or more containers, with each pod representing an instance of the application. A pod is the smallest unit that K8s can manage. All containers in the pod share the same network name space.

We rely on K8s to orchestrate the infrastructure that the cloud-native router needs to operate. However, we do not supply K8s installation or management instructions in this documentation. See https://kubernetes.io for Kubernetes documentation. Currently, Juniper Cloud-Native Router requires that the K8s cluster be a standalone cluster, meaning that the K8s master and worker functions both run on a single node.

Juniper Cloud-Native Router Components

Juniper Cloud-Native Router Controller
The JCNR-Controller (cRPD) is the control-plane part of the Juniper Cloud-Native Router solution. You use the controller to communicate with the other elements of the cloud-native router. Configuration, policies and rules that you set on the controller at deploy time are communicated to other components, primarily the JCNR-vRouter-agent and JCNR-vRouter for implementation.

For example, access control lists (ACLs) are supported on JCNR-Controller to configure L2 access lists with deny rules. JCNR-controller sends the configuration information to the JCNR-vRouter through the JCNR-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 JCNR-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 JCNR-vRouter without passing through the kernel.
- Support for configuration of VLAN-tagged sub-interfaces on physical function (PF), virtual function (VF), virtio, access, and trunk interfaces managed by the DPDK-enabled JCNR-vRouter.
- Supports configuration of bridge domains

Juniper Cloud-Native Router-vRouter
JCNR-vRouter is an alternative to the Linux bridge or the Open vSwitch (OVS) module in the Linux kernel. The pod which houses the JCNR-vRouter container also houses the JCNR-vRouter agent container. JCNR-vRouter functions to:

- Perform L2 forwarding
- Perform L2 rate-limiting
- Allows the use of DPDK-based forwarding
- Enforce L2 access control lists (ACLs)

JCNR-Container Network Interface (JCNR-CNI)
JCNR-CNI is a new CNI developed by Juniper to handle Juniper-developed Pods like JCNR-vRouter agent and JCNR-vRouter agent DPDK, along with DPDK application Pods and the cloud-native router controller. JCNR-CNI is a kubernetes CNI plugin installed on each node to provision network interfaces for application pods. During pod creation, K8s delegates Pod interface creation and configuration to JCNR-CNI. JCNR-CNI interacts with JCNR control-plane and JCNR-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 K8s and cloud-native router components. JCNR-CNI works with the Multus CNI to add and configure Pod interfaces.

JCNR-CNI provides the following functionality:

- Manages the networking tasks in K8s pods such as assigning IP addresses, allocating MAC addresses, and setting up interfaces between the Pod and host in a K8s cluster
- Applies L2 ACLs. The policies are sent to JCNR-vRouter for applying in the data plane
- Acts on Pod events such as add and delete
- Generates cRPD configuration

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 and can be accessed from that file.

System Resource Requirements

Read this section to understand the Linux host requirements for Juniper Cloud-Native Router.

The following tables list the host system requirements on page 6 for installing cloud-native router in L2 mode, cloud-native router resource requirements on page 8, and other miscellaneous requirements on page 9.

Table 1: Cloud-Native Router Host System Requirements

Component	Release 22.2		Release 22.3	
	Value/ Version	Notes	Value/Version	Notes
CPU	Intel x86	The tested CPU is Intel Xeon Gold 6212U 24-core @2.4 GHz	Intel x86	The tested CPU is Intel Xeon Gold 6212U 24-core @2.4 GHz
Host OS	RedHat Enterprise Linux	Version 8.4, 8.5, 8.6	RedHat Enterprise Linux	Version 8.4, 8.5, 8.6

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

Component	Release 22.2		Release 22.3	
	Value/ Version	Notes	Value/Version	Notes
Kernel Version	4.18.X	The tested kernel version is 4.18.0-305.rt7.72.el8.x86_64	4.18.X	The tested kernel version is 4.18.0-305.rt7.72.el8.x86_64
NIC	Intel E810 with Firmware 3.20 0x8000d853 1.3146.0		• Intel E810 with Firmware 3.20 0x8000d853 1.3146.0 • Intel XL710 with Firmware 8.60	
IAVF driver	Version 4.4.2		Version 4.4.2	
ICE_COMMS	Version 1.3.35.0		Version 1.3.35.0	
ICE	Version 1.8.3.1.2		Version 1.8.3.1.2	ICE driver is used only with the Intel E810 NIC
i40e			Version 2.18.9	i40e driver is used only with the Intel XL710 NIC
Kubernetes (K8s)	Version 1.22.2	The tested K8s version is 1.22.4, although 1.22.2 will also work. **NOTE**: The K8s cluster must be a standalone/all-in-one cluster	Version 1.22.2	The tested K8s version is 1.22.4, although 1.22.2 will also work. **NOTE**: The K8s cluster must be a standalone/all-in-one cluster

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

Component	Release 22.2		Release 22.3	
	Value/ Version	Notes	Value/Version	Notes
Calico	Version 3.22.0		Version 3.22.0	
Multus	Version 3.8		Version 3.8	
Helm	3.9.x		3.9.x	
Container-RT	Docker CE 20.10.11		Docker CE 20.10.11	

Table 2: Cloud-Native Router Resource Requirements

Resource	Releases 22.2 and 22.3	
	Value	Usage Notes
Data plane forwarding cores	2 physical cores (2p)	
Service/Control Cores	0	
UIO Driver	VFIO-PCI	
Hugepages (1G)	4 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"`
JCNR Controller cores	.5	
JCNR vRouter Agent cores	.5	

Table 3: Miscellaneous Requirements

Cloud-Native Router Release	Requirement
22.2 and 22.3	Enable VLAN driver at system boot
	Enable VFIO-PCI driver at system boot
	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"`.
	Disable Spoofcheck on VFs allocated to JCNR. For example: `ip link set <interfacename> vf 1 spoofcheck off`.
	Set trust on VFs allocated to JCNR. For example: `ip link set <interfacename> vf 1 trust on`

2

CHAPTER

Deploy Juniper Cloud-Native Router

Install Juniper Cloud-Native Router

SUMMARY

The Juniper Cloud-Native Router (JCNR) uses the the JCNR-Controller (cRPD-based control plane) and JCNR-CNI to provide control plane capabilities and 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

The JCNR-Controller (cRPD) is an initialization container that provides control plane functionality for the cloud-native router. The control plane is responsible for provisioning of the workload and fabric interfaces used in Juniper Cloud-Native Router. It also manages communication with the vRouter-agent and the vRouter itself over a gRPC connection.

The JCNR-CNI is the container network interface that Juniper Cloud-Native Router uses to communicate with physical interfaces on the server and pod and container network interfaces within the installation.

The Juniper Cloud-Native Router Virtual Router (vRouter) is a container application set that provides advanced forwarding plane functionality. It extends the network from the physical routers and switches into a virtual overlay network hosted in the virtualized servers. The Data Plane Development Kit (DPDK) enables the vRouter to process more packets per second than is possible when the vRouter runs as a kernel module.

The Syslog-NG is a container application that allows Juniper Cloud-Native Router to provide notifications to users about events that occur in the cloud-native router deployment.

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.

As mentioned in the "System Resource Requirements" on page 6, the Helm package manager for Kubernetes must be installed prior to installing Juniper Cloud-Native Router components.

NOTE: We do not provide a specific path into which you must download the package and install the software. Because, of this you can copy the commands shown throughout this document and paste them into the CLI of your server.

The high-level overview of Juniper Cloud-Native Router installation is:

1. Download the software installation package (tarball)
2. Expand the tarball
3. Change directory to Juniper_Cloud_Native_Router_<release number>
4. Load the image files into Docker
5. Enter the root password for your host server and your Juniper Cloud-Native Router
6. Apply the **secrets/jcnr-secrets.yaml** to the K8s system
7. Edit the values.yaml file to suit the needs of your installation
8. Install the Juniper Cloud-Native Router

Each high-level procedure listed above is detailed below,

1. Download the tarball, **Juniper_Cloud_Native_Router_<release-number>.tgz**, to the directory of your choice.

   How you get the tarball into a writeable directory on your server is up to you. You must perform the file transfer in binary mode so the compressed tar file will expand properly.

2. Expand the file `Juniper_Cloud_Native_Router_<release-number>.tgz`.

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

```
Juniper_Cloud_Native_Router_22.3/
Juniper_Cloud_Native_Router_22.3/secrets/
Juniper_Cloud_Native_Router_22.3/secrets/jcnr-secrets.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/Chart.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/values.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/
```

```
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/.helmignore
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/Chart.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/templates/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/templates/_helpers.tpl
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/templates/
jcnrvrouter_cleanup.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/templates/vrouter.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/values.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-vrouter/README.md
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/.helmignore
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/Chart.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/files/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/files/syslog-ng.conf
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/templates/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/templates/_helpers.tpl
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/templates/syslog.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/templates/syslog-
config.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/syslog-ng/values.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/.helmignore
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/Chart.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/files/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/files/jcnr-cni-config.tmpl
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/templates/
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/templates/_helpers.tpl
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/templates/jcnr-config.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/templates/jcnr_cleanup.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/templates/jcnr.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/templates/jcnr-nad.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/values.yaml
Juniper_Cloud_Native_Router_22.3/helm_charts/jcnr/charts/jcnr-cni/README.md
Juniper_Cloud_Native_Router_22.3/contrail-tools/
Juniper_Cloud_Native_Router_22.3/contrail-tools/contrail-tools.yaml
Juniper_Cloud_Native_Router_22.3/images/
Juniper_Cloud_Native_Router_22.3/images/jcnr-cni-images.tar.gz
Juniper_Cloud_Native_Router_22.3/images/jcnr-vrouter-images.tar.gz
Juniper_Cloud_Native_Router_22.3/images/syslog-ng-images.tar.gz
Juniper_Cloud_Native_Router_22.3/README.md
```

3. Change directory to Juniper_Cloud_Native_Router_22.3

```
cd Juniper_Cloud_Native_Router_22.3
```

> **NOTE**: All remaining steps in the installation assume that your current working directory is now **Juniper_Cloud_Native_Router_22.3**.

4. Load the image files, **jcnr-cni-images.tar.gz**, **jcnr-vrouter-images.tar.gz**, and **syslog-ng-images.tar.gz** into docker. The image files are located in the **Juniper_Cloud_Native_Router_22.3/images** directory relative to where you expanded the tarball in the previous step.

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

```
94c4181ae7dd: Loading layer [==================================================>]  524.2MB/
524.2MB
Loaded image: svl-artifactory.juniper.net/junos-docker-local/warthog/crpd:22.3R1-S1.5
86441b6792e3: Loading layer [==================================================>]  160.3kB/
160.3kB
2f858df19dda: Loading layer [==================================================>]  26.74MB/
26.74MB
dc5b9d2f0f0a: Loading layer [==================================================>]   7.68kB/
7.68kB
Loaded image: svl-artifactory.juniper.net/junos-docker-local/warthog/jcnr-cni:20220810-f753972
67e3ffed6327: Loading layer [==================================================>]  11.26kB/
11.26kB
Loaded image: svl-artifactory.juniper.net/junos-docker-local/warthog/crpdconfig-generator:v3
Loaded image: svl-artifactory.juniper.net/atom_virtual_docker/busybox:latest
```

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

```
50244b5caf0a: Loading layer [==================================================>]  3.584kB/
3.584kB
4793f35d3ae7: Loading layer [==================================================>]  5.632kB/
5.632kB
f1697a784d3d: Loading layer [==================================================>]  3.584kB/
3.584kB
```

```
9d6c27fd1364: Loading layer [==================================================>]  28.79MB/
28.79MB
877da3dd69a5: Loading layer [==================================================>]  11.26kB/
11.26kB
f3519070976e: Loading layer [==================================================>]  1.396MB/
1.396MB
145f8619ed40: Loading layer [==================================================>]  183.5MB/
183.5MB
d4fe4ae73ff1: Loading layer [==================================================>]  6.812MB/
6.812MB
9a9e214f9045: Loading layer [==================================================>]  2.467MB/
2.467MB
2c026bac5448: Loading layer [==================================================>]  41.59MB/
41.59MB
8f873ca42faf: Loading layer [==================================================>]  40.13MB/
40.13MB
b6e9fea633a7: Loading layer [==================================================>]  72.19kB/
72.19kB
19234c4cbb31: Loading layer [==================================================>]  498.2kB/
498.2kB
f9c52ee9be26: Loading layer [==================================================>]  18.66MB/
18.66MB
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-vrouter-
agent:JCNR-22.3-6
8d7366c22fd8: Loading layer [==================================================>]  3.697MB/
3.697MB
a93413564615: Loading layer [==================================================>]  18.29MB/
18.29MB
415be476c298: Loading layer [==================================================>]  1.352MB/
1.352MB
e83d4114481d: Loading layer [==================================================>]  2.365MB/
2.365MB
623e9ce88f39: Loading layer [==================================================>]   2.81MB/
2.81MB
f4e7db2826f6: Loading layer [==================================================>]  54.39MB/
54.39MB
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-vrouter-
kernel-init-dpdk:JCNR-22.3-6
4aa40b94fdb4: Loading layer [==================================================>]  5.632kB/
5.632kB
2079da3dd3ea: Loading layer [==================================================>]  250.6MB/
250.6MB
5f6b6a83bbc2: Loading layer [==================================================>]  22.02kB/
```

```
22.02kB
3c479d39cdd2: Loading layer [==================================================>]  13.82kB/
13.82kB
a8e86ba6a002: Loading layer [==================================================>]  9.216kB/
9.216kB
ce903f8e71cc: Loading layer [==================================================>]  25.03MB/
25.03MB
8b94a98d6508: Loading layer [==================================================>]  372.5MB/
372.5MB
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-
tools:JCNR-22.3-6
Loaded image: svl-artifactory.juniper.net/atom_virtual_docker/busybox:latest
Loaded image: svl-artifactory.juniper.net/junos-docker-local/warthog/busybox:latest
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-telemetry-
exporter:JCNR-22.3-6
cfd97936a580: Loading layer [==================================================>]  1.455MB/
1.455MB
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/busybox:latest
1e7fbcfd6526: Loading layer [==================================================>]  6.144kB/
6.144kB
1b2e64d61760: Loading layer [==================================================>]  182.2MB/
182.2MB
c58c4d0e394a: Loading layer [==================================================>]  68.61kB/
68.61kB
ae3bac1d08f4: Loading layer [==================================================>]  4.608kB/
4.608kB
173cf86d714f: Loading layer [==================================================>]  720.9kB/
720.9kB
7c5fc69220bb: Loading layer [==================================================>]  4.608kB/
4.608kB
fc23189a25c0: Loading layer [==================================================>]  4.608kB/
4.608kB
9e4ba1a88748: Loading layer [==================================================>]  44.09MB/
44.09MB
999a87d50c2c: Loading layer [==================================================>]  54.22MB/
54.22MB
b7247dc2bdc4: Loading layer [==================================================>]  478.7kB/
478.7kB
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-vrouter-
dpdk:JCNR-22.3-6
e94ef981fc21: Loading layer [==================================================>]  63.51MB/
63.51MB
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-k8s-
```

```
crdloader:JCNR-22.3-6
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-k8s-
applier:JCNR-22.3-6
a5af639475d7: Loading layer [==================================================>]  59.58MB/
59.58MB
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-k8s-
deployer:JCNR-22.3-6
da2cd5786adc: Loading layer [==================================================>]  62.68MB/
62.68MB
Loaded image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/contrail-
init:JCNR-22.3-6
```

```
docker load -i images/syslog-ng-images.tar.gz
```

```
Loaded image: svl-artifactory.juniper.net/contrail-docker/syslog-ng:v6
```

5. 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.

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

   ```
   base64 rootPasswordFile
   ```

   The output is a single line of random-looking text similar to:

   ```
   UGFzc3cwcmQhCg==
   ```

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

   ```
   base64 licenseFile
   ```

   The output is a long single line of random-looking text similar to:

   ```
   VGhpcyBpcyBhIHJlYWxseSBtdWNoIGxvbmdlciB0ZXh0IGZpbGUgdGhhdCBpbmNsdWRlcyBsaWNlbnNlIGluZm9ybWF0aW9uCkFTREZERktERk
   tIQUxHS0hiYW9qa2hkZmFzZGZOS0FTREdOR0FKYWRzZmxodmFibmRzZmdramh2Ym5ramFzZnVxYmF1amgyMDEwdGIydDQweGtqYjR3eTB1dmRx
   d3J2MGl3aGV0Ymd1YnMwcWRqZmhkc2tqdmJkc2Zramhkdmfkznni02d2a2pzZGI7aWRzamc7ZmFzZGhma2pkc2J2YWlzdWRmZ3dFWUlUR1ZCMz
   lWRVlCVjM0OVVHQlZHQlFVOUFXR1ZJQkVSV0c5VUJWV0U5Rwo=
   ```

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

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

6. Apply the **secrets/jcnr-secrets.yaml** to the K8s system

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

7. Edit the **helm_charts/jcnr/values.yaml** file.

   You must customize the Helm chart for the Juniper Cloud-Native Router installation:

   - Choose fabric interfaces–Use interface names from your host system
   - Create the VLAN id list for trunk interfaces–Use VLAN ids that fit in your network
   - Choose a fabric workload interface-Use interface names from your host system
   - Set the VLAN id for traffic on the workload interface
   - Set the severity level for JCNR-vRouter logging

     > **NOTE**: Leave the log_level set to INFO unless instructed to change it by JTAC.

   - Set the cpu core mask–physical cores, logical cores
   - Choose the fabric interface–Use interface names from your host system
   - Choose a workload interface–Use interface names from your host system
   - Set a rate limit for broadcast and multicast traffic in bytes per second
   - Set a writeable directory location for syslog-ng to store notifications
   - (Optional) If you specify a bond interface as your `fabricInterface:`, provide `slaveInterface` names from your system under the `bondInterfaceConfigs:` section.
   - By default `restoreInterface` is set to false. With this setting when vrouter pod crashes or is deleted the interfaces are not restored back to host.

NOTE: If you are using the Intel XL710 NIC, you must set `ddp=false` in the **values.yaml**

See "Sample Configuration Files" on page 35 for a commented example of the default **helm_charts/jcnr/values.yaml** file.

8. Deploy the Juniper Cloud-Native Router using Helm

```
helm install jcnr .
```

```
NAME: jcnr
LAST DEPLOYED: Mon Aug  15 14:29:41 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

9. Confirm Juniper Cloud-Native Router Deployment

```
helm ls
```

```
NAME    NAMESPACE       REVISION        UPDATED
STATUS          CHART           APP VERSION
jcnr    default         1               2022-08-17 18:51:18.472130634 -0700 PDT
deployed        jcnr-22.3.0     22.3.0
```

Verify Operation of Containers

This task allows you to confirm that the Juniper Cloud-Native Router Pods are running.

1. `kubectl get pods -A`

The output of the `kubectl` command shows all of the pods in the K8s cluster in all namespaces. Successful deployment means that all pods display that they 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-7b5dd699b9-nd7xf     1/1     Running
0             41m
contrail          contrail-vrouter-masters-dfxgm            3/3     Running
0             41m
default           delete-crpd-dirs--1-6jmxz                 0/1     Completed
0             43m
default           delete-vrouter-dirs--1-645dt              0/1     Completed
0             43m
kube-system       calico-kube-controllers-57b9767bdb-5wbj6  1/1     Running     2 (92d
ago)   129d
kube-system       calico-node-j4m5b                         1/1     Running     2 (92d
ago)   129d
kube-system       coredns-8474476ff8-fpw78                  1/1     Running     2 (92d
ago)   129d
kube-system       dns-autoscaler-7f76f4dd6-q5vdp            1/1     Running     2 (92d
ago)   129d
kube-system       kube-apiserver-5a5s5-node2                1/1     Running     3 (92d
ago)   129d
kube-system       kube-controller-manager-5a5s5-node2       1/1     Running     4 (92d
ago)   129d
kube-system       kube-crpd-worker-ds-8tnf7                 1/1     Running
0             41m
kube-system       kube-multus-ds-amd64-4zm5k                1/1     Running     2 (92d
ago)   129d
kube-system       kube-proxy-l6xm8                          1/1     Running     2 (92d
ago)   129d
kube-system       kube-scheduler-5a5s5-node2                1/1     Running     4 (92d
ago)   129d
kube-system       nodelocaldns-6kwg5                        1/1     Running     2 (92d
ago)   129d
kube-system       syslog-ng-54749b7b77-v24hq                1/1     Running
0             41m
```

2. kubectl get ds -A

   Use the kubectl get ds -A command to get a list of daemonset containers.

```
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
node-role.kubernetes.io/master=   43m
kube-system   calico-node                 1         1         1       1            1
kubernetes.io/os=linux            129d
kube-system   kube-crpd-worker-ds         1         1         1       1            1
<none>                            43m
kube-system   kube-multus-ds-amd64        1         1         1       1            1
kubernetes.io/arch=amd64          129d
kube-system   kube-proxy                  1         1         1       1            1
kubernetes.io/os=linux            129d
kube-system   nodelocaldns                1         1         1       1            1
kubernetes.io/os=linux            129d
```

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

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 4: Investigate Deployment Issues

Potential issue	What to check	Related Commands
Image not found	Check if registry is accessible, image tags are correct	• `kubectl -n kube-system describe pod <crpd-pod-name>`
Initialization errors	Check if jcnr-secrets is loaded and has a valid license key	`cat /var/run/jcnr/juniper.conf` Confirm that root password and license key are present

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 describe pod <crpd-pod-name>`
		• See "Access the Cloud-Native Router CLIs" on page 26 to enter the cRPD CLI and run the following command:
		`show krt state channel vrouter`
		• `cat /var/run/jcnr/juniper.conf`

Table 4: Investigate Deployment Issues *(Continued)*

Potential issue	What to check	Related Commands
vRouter Pod in CrashLoopBackOff state	Check the contail-k8s-deployer pod for errors	`kubectl logs contrail-k8s-deployer-<pod-hash> -n contrail-deploy`

View 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. You can view this configuration file to see the details of the cRPD configuration. To view the cRPD configuration:

1. Navigate to the **/var/run/jcnr** folder to access the configuration file details.

   root@server:/var/run/jcnr#`ls`

   ```
   config  containers  juniper.conf  jcnr-crpd-pod.conf
   ```

2. View the contents of the configuration file.

   root@server:/var/run/jcnr#`vi juniper.conf`

View Log Files

In this topic, we use the default log_path directory, **/var/log/jcnr/**, and the default syslog_notifications directory, **/var/log/jcnr/jcnr-notifications.json**. 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@host: /var/log/jcnr]# ls
```

```
contrail-vrouter-agent.log  contrail-vrouter-dpdk-init.log  contrail-vrouter-dpdk.log  vrouter-
kernel-init.log
calico                  containers           cloud-init.log          contrail             jcnr-
cni.log
cloud-init-output.log   crpd           pods       jcnr-notifications.json
```

3

CHAPTER

Post Deployment

Manage Cloud-Native Router Controller and Cloud-Native Router vRouter

SUMMARY

This topic contains instructions for how to access the cloud-native router CLIs, how to run operational commands in cRPD and vRouter containers, and how to remove cloud-native router.

IN THIS SECTION

Access the Cloud-Native Router CLIs

You can access the cloud-native router's CLI to monitor the router's status and to make configuration changes. In this section we provide the commands that you use to access the cRPD and vRouter CLIs and provide some examples of show commands.

Because the cloud-native router controller element runs as a Pod in a Kubernetes (K8s) cluster, you must use K8s commands to access the CLI. We provide an example below. We do not provide specific directory paths in our examples so you can copy and paste the commands into your server.

Access the Cloud-Native Router Controller (cRPD) CLI

In this example we list all of the K8s Pods running on the K8s host server. We use that output to identify the cRPD Pod that hosts the cloud-native router controller container. We then connect to the CLI of the cloud-native router controller and run some show commands.

List the K8s Pods Running in the Cluster

```
kubectl get pods -A
```

```
NAMESPACE         NAME                                    READY   STATUS
RESTARTS      AGE
contrail-deploy   contrail-k8s-deployer-7b5dd699b9-nd7xf   1/1     Running
0             41m
contrail          contrail-vrouter-masters-dfxgm          3/3     Running
0             41m
```

```
default           delete-crpd-dirs--1-6jmxz                  0/1     Completed
0             43m
default           delete-vrouter-dirs--1-645dt               0/1     Completed
0             43m
kube-system       calico-kube-controllers-57b9767bdb-5wbj6   1/1     Running     2 (92d
ago)   129d
kube-system       calico-node-j4m5b                          1/1     Running     2 (92d
ago)   129d
kube-system       coredns-8474476ff8-fpw78                   1/1     Running     2 (92d
ago)   129d
kube-system       dns-autoscaler-7f76f4dd6-q5vdp             1/1     Running     2 (92d
ago)   129d
kube-system       kube-apiserver-5a5s5-node2                 1/1     Running     3 (92d
ago)   129d
kube-system       kube-controller-manager-5a5s5-node2        1/1     Running     4 (92d
ago)   129d
kube-system       kube-crpd-worker-ds-8tnf7                  1/1     Running
0             41m
kube-system       kube-multus-ds-amd64-4zm5k                 1/1     Running     2 (92d
ago)   129d
kube-system       kube-proxy-l6xm8                           1/1     Running     2 (92d
ago)   129d
kube-system       kube-scheduler-5a5s5-node2                 1/1     Running     4 (92d
ago)   129d
kube-system       nodelocaldns-6kwg5                         1/1     Running     2 (92d
ago)   129d
kube-system       syslog-ng-54749b7b77-v24hq                 1/1     Running
0             41m
```

The only Pod that has cRPD in its name is the **kube-crpd-worker-ds-npbjq**. Thus, this is the name of the Pod we will use to access the cRPD CLI.

Connect to the cRPD CLI

The kubectl command that allows access to the controller's CLI has the following form:

```
kubectl exec -n <namespace> -it <cRPD worker Pod name> -- bash
```

In practice, you substitute values from your system for the values contained between angle brackets (<>). For example:

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

The result of the above command should appear similar to:

```
===>
           Containerized Routing Protocols Daemon (CRPD)
 Copyright (C) 2020-2021, Juniper Networks, Inc. All rights reserved.
                                                                    <===
root@ix-jcnr-01:/#
```

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

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

```
root@jcnr-01>
```

Example Show Commands

In the following examples, we remove the prompt, **root@jcnr-01>**, so you can copy and paste the commands into your system without editing them.

```
show interfaces terse
```

```
__crpd-brd1: flags=67<UP,BROADCAST,RUNNING>  mtu 1500
        inet6 fe80::205f:39ff:fe19:87b7  prefixlen 64  scopeid 0x20<link>
        ether 22:5f:39:19:87:b7  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7  bytes 746 (746.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

cali502530ac57f: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1480
        inet6 fe80::ecee:eeff:feee:eeee  prefixlen 64  scopeid 0x20<link>
        ether ee:ee:ee:ee:ee:ee  txqueuelen 0  (Ethernet)
        RX packets 9530538  bytes 816771272 (816.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11502794  bytes 11091296232 (11.0 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
caliae604977c78: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1480
        inet6 fe80::ecee:eeff:feee:eeee  prefixlen 64  scopeid 0x20<link>
        ether ee:ee:ee:ee:ee:ee  txqueuelen 0  (Ethernet)
        RX packets 10120320  bytes 904273274 (904.2 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9242684  bytes 841165346 (841.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:b9:ad:64:ad  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.87.3.138  netmask 255.255.255.128  broadcast 10.87.3.255
        inet6 fe80::3eec:efff:fe4e:145c  prefixlen 64  scopeid 0x20<link>
        ether 3c:ec:ef:4e:14:5c  txqueuelen 1000  (Ethernet)
        RX packets 10432410  bytes 10076907508 (10.0 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3444445  bytes 3877176824 (3.8 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device memory 0xaae20000-aae3ffff

eno2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 3c:ec:ef:4e:14:5d  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device memory 0xaae00000-aae1ffff

enp59s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 40:a6:b7:2a:86:78  txqueuelen 1000  (Ethernet)
        RX packets 25596  bytes 5412132 (5.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 660 (660.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp59s0f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 40:a6:b7:2a:86:79  txqueuelen 1000  (Ethernet)
```

```
        RX packets 554  bytes 116931 (116.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7  bytes 770 (770.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp59s0f1v1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 72:e4:ae:81:4a:b3  txqueuelen 1000  (Ethernet)
        RX packets 7  bytes 2048 (2.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 612  bytes 107701 (107.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
        inet 169.254.93.210  netmask 255.255.255.255  broadcast 0.0.0.0
        inet6 fe80::58b2:7fff:fea9:3adb  prefixlen 64  scopeid 0x20<link>
        ether 5a:b2:7f:a9:3a:db  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 70 (70.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

gre0: flags=193<UP,RUNNING,NOARP>  mtu 1476
        unspec 00-00-00-00-30-30-30-3A-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ip6tnl0: flags=193<UP,RUNNING,NOARP>  mtu 1452
        inet6 fe80::d4c4:b5ff:fede:df84  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

irb: flags=67<UP,BROADCAST,RUNNING>  mtu 1500
        inet6 fe80::f4a0:c8ff:fee6:a28c  prefixlen 64  scopeid 0x20<link>
        ether f6:a0:c8:e6:a2:8c  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4885871889  bytes 578917760800 (578.9 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4885871889  bytes 578917760800 (578.9 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lsi: flags=67<UP,BROADCAST,RUNNING>  mtu 1500
        inet6 fe80::4fd:d0ff:fe4e:943b  prefixlen 64  scopeid 0x20<link>
        ether 06:fd:d0:4e:94:3b  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 440 (440.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

sit0: flags=193<UP,RUNNING,NOARP>  mtu 1480
        inet6 ::127.0.0.1  prefixlen 96  scopeid 0x90<compat,host>
        inet6 ::172.17.0.1  prefixlen 96  scopeid 0x80<compat,global>
        inet6 ::169.254.93.210  prefixlen 96  scopeid 0x80<compat,global>
        inet6 ::10.87.3.138  prefixlen 96  scopeid 0x80<compat,global>
        sit  txqueuelen 1000  (IPv6-in-IPv4)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tunl0: flags=193<UP,RUNNING,NOARP>  mtu 1480
        inet 10.233.90.0  netmask 255.255.255.255
        tunnel   txqueuelen 1000  (IPIP Tunnel)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

```
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
show configuration routing-instances
```

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

Access the Cloud-Native Router vRouter CLI

In this example we list all of the K8s Pods running on the K8s host server. We use that output to identify the vRouter Pod that hosts the cloud-native router vrouter-agent container. We then connect to the CLI of the vRouter-agent and run a show command to list the available interfaces.

List the K8s Pods Running in the Cluster

```
kubectl get pods -n contrail
```

```
NAME                              READY   STATUS    RESTARTS   AGE
contrail-vrouter-masters-dfxgm    3/3     Running   0          79m
```

Connect to the Cloud-Native Router vRouter CLI

The kubectl command that allows access to the controller's CLI has the following form:

```
kubectl exec -n contrail -it <contrail-vrouter-masters-pod> -- bash
```

In practice, you substitute values from your system for the values contained between angle brackets (<>). For example:

```
kubectl exec -n contrail -it contrail-vrouter-masters-xnwwp -- bash
```

The output of this command should look similar to:

```
root@jcnr-01:/#
```

At this point, you have connected to the vRouter's CLI. You can run commands in the CLI to learn about the state of the vRouter. For example, the command shown below allows you to see which interfaces are present on the vRouter.

```
root@jcnr-01:/# vif --list
```

```
Vrouter Operation Mode: PureL2
Vrouter Interface Table

Flags: P=Policy, X=Cross Connect, S=Service Chain, Mr=Receive Mirror
       Mt=Transmit Mirror, Tc=Transmit Checksum Offload, L3=Layer 3, L2=Layer 2
       D=DHCP, Vp=Vhost Physical, Pr=Promiscuous, Vnt=Native Vlan Tagged
```

```
       Mnp=No MAC Proxy, Dpdk=DPDK PMD Interface, Rfl=Receive Filtering Offload,
Mon=Interface is Monitored
       Uuf=Unknown Unicast Flood, Vof=VLAN insert/strip offload, Df=Drop New Flows, L=MAC
Learning Enabled
       Proxy=MAC Requests Proxied Always, Er=Etree Root, Mn=Mirror without Vlan Tag,
HbsL=HBS Left Intf
       HbsR=HBS Right Intf, Ig=Igmp Trap Enabled, Ml=MAC-IP Learning Enabled, Me=Multicast
Enabled

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

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

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

Remove the Juniper Cloud-Native Router

We do not provide specific directory names for the commands in this topic. This allows you to copy and paste the commands from this document onto your server.

Uninstall the Juniper Cloud-Native Router.

```
helm uninstall jcnr
```

Sample Configuration Files

Read this section to find sample YAML configuration files for use in deploying Juniper Cloud-Native Router. These YAML files control the features and functions available to cloud-native router by affecting the deployment instructions. YAML files for workload configuration are also included. The workload configuration files control the workload functions.

We've included the following sample configuration files:

- **Juniper Cloud-Native Router Main Configuration FIle**
- • main "values.yaml" on page 36file
- **Juniper Cloud-Native Router vRouter-Specific Configuration File**
  "jcnr-vrouter specific values.yaml file" on page 37
- **Juniper Cloud-Native Router JCNR-CNI-Specific Configuration File**
  "jcnr-cni specific values.yaml file" on page 41
- **Workload Configuration Files**
- • "nad-dpdk_trunk_vlan_3002.yaml" on page 43
  - " nad-kernel_access_vlan_3001.yaml" on page 44
  - " nad-odu-bd3003-sub.yaml" on page 45
  - "nad-odu-bd3004-sub.yaml" on page 46
  - "odu-virtio-subinterface.yaml" on page 47
  - "pod-dpdk-trunk-vlan3002.yaml " on page 49
  - "pod-kernel-access-vlan-3001.yaml" on page 50

Use these files to understand the configuration options available for deployment of Juniper Cloud-Native Router. The workload configuration files display how you can configure trunk and access interfaces and configure various VLANs for each type. Each of the files contain comments that start with a hash mark (#) and are highlighted in **bold** in these examples.

- values.yaml

  This is the main **values.yaml file**. There are 3 other values.yaml files supplied in the TAR file. 1 **values.yaml** for each of the installation components: **jcnr-cni**, **jcnr-vrouter**, and **syslog-ng**.

  If there are conflicting settings between the individual **values.yaml** files and the main **values.yaml** file, the settings in the main **values.yaml** file take precedence.

```
##################################################################
#                Common Configuration (global vars)              #
##################################################################
global:
  registry: svl-artifactory.juniper.net/
  # uncomment below if all images are available in the same path; it will
  # take precedence over "repository" paths under "common" section below
  #repository: path/to/allimages/
  common:
    vrouter:
      repository: atom-docker/cn2/bazel-build/dev/
      tag: JCNR-22.3-6
    crpd:
      repository: junos-docker-local/warthog/
      tag: 22.3R1.8
    jcnrcni:
      repository: junos-docker-local/warthog/
      tag: 20220918-4adf886

  # 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"

  # fabricInterface: NGDU or tor side interface, expected all types
```

```
  # of traffic; interface_mode is always trunk for this mode
  fabricInterface:
  - bond0:
      interface_mode: trunk
      vlan-id-list: [100, 200, 300, 700-705]

  # fabricWorkloadInterface: RU side interfaces, expected traffic is only
  # management/control traffic; interface mode is always access for this mode
  fabricWorkloadInterface:
  - enp59s0f1v0:
      interface_mode: access
      vlan-id-list: [700]

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

  # bond interface configurations
  bondInterfaceConfigs:
    - name: "bond0"
      mode: 1             # ACTIVE_BACKUP MODE -- This is the only supported mode
      slaveInterfaces:
      - "enp59s0f0v0"
      - "enp59s0f0v1"

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

  # vrouter fwd core mask
  cpu_core_mask: "2,3"

  # rate limit for broadcast/multicast traffic on fabric interfaces in bytes per second
  fabricBMCastRateLimit: 0

  # Set ddp to true to enable Dynamic Device Personalization (DDP)
  # It provides datapath optimization at NIC for traffic like GTPU, SCTP etc.
  ddp: true #set to false if you use the Intel XL710 NIC
```

- jcnr-vrouter specific `values.yaml`

This **values.yaml** file is specific to the jcnr-vrouter Pod. It is located under the **Juniper_Cloud_Native_Router_<release-number>/helm_charts/jcnr/charts/jcnr-vrouter** directory. If you enter any values in this file that conflict with values in the main **values.yaml** file, the values in the main **values.yaml** file take precedence.

```
# # This is a YAML-formatted file.
# # Declare variables to be passed into your templates.

common:
  registry: svl-artifactory.juniper.net/
  repository: atom-docker/cn2/bazel-build/dev/

# anchor tag for vrouter container images
vrouter-tag: &vrouter_tag JCNR-22.3-6

contrail_init:
  image: contrail-init
  tag: *vrouter_tag
  pullPolicy: IfNotPresent

contrail_vrouter_kernel_init_dpdk:
  image: contrail-vrouter-kernel-init-dpdk
  tag: *vrouter_tag
  pullPolicy: IfNotPresent

contrail_vrouter_agent:
  image: contrail-vrouter-agent
  tag: *vrouter_tag
  pullPolicy: IfNotPresent

contrail_vrouter_agent_dpdk:
  image: contrail-vrouter-dpdk
  tag: *vrouter_tag
  pullPolicy: IfNotPresent
  resources:
    limits:
      memory: 4Gi
      hugepages-1Gi: 4Gi          # Hugepages must be enabled with default size as 1G;
minimum 4Gi to be used
    requests:
      memory: 4Gi
      hugepages-1Gi: 4Gi
```

```
contrail_vrouter_telemetry_exporter:
  image: contrail-telemetry-exporter
  tag: *vrouter_tag
  pullPolicy: IfNotPresent

contrail_k8s_deployer:
  image: contrail-k8s-deployer
  tag: *vrouter_tag
  pullPolicy: IfNotPresent

contrail_k8s_crdloader:
  image: contrail-k8s-crdloader
  tag: *vrouter_tag
  pullPolicy: IfNotPresent

contrail_k8s_applier:
  image: contrail-k8s-applier
  tag: *vrouter_tag
  pullPolicy: IfNotPresent

busyBox:
  image: busybox
  tag:  "latest"
  pullPolicy: IfNotPresent

vrouter_name: master

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

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

vrouter_log_path: "/var/log/jcnr/"

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

dpdkCommandAdditionalArgs: "--yield_option 0"

# Set ddp to true to enable Dynamic Device Personalization (DDP)
# It provides datapath optimization at NIC for traffic like GTPU, SCTP etc.
```

```
ddp: true

# vrouter fwd core mask
cpu_core_mask: "2,3"

# vrouter service thread mask
service_core_mask: ""

# vrouter control thread mask
dpdk_ctrl_thread_mask: ""

#
dpdk_mem_per_socket: "1024"

# L3 disabled for switching mode
jcnr_mode: "l2_only"

# global Mac table size - We recommend leaving this at the default value
mac_table_size: "10240"

# timeout (seconds) for aging Mac table entries (S)
mac_table_ageout: 60

# parameters for vRouter livenessProbe
livenessProbe:
  initialDelaySeconds: 10
  periodSeconds: 20
  timeoutSeconds: 5
  failureThreshold: 3
  successThreshold: 1

# parameters for vRouter startupProbe
startupProbe:
  initialDelaySeconds: 10
  periodSeconds: 20
  timeoutSeconds: 5
  failureThreshold: 3
  successThreshold: 1

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

```
# tor side interface, expected all types of traffic
fabricInterface:
- enp4s0f0vf0
- bond0

# RU side interfaces, expected traffic is only management/control traffic
fabricWorkloadInterface:
- enp4s0f1vf0

# bond interface configurations
bondInterfaceConfigs:
  - name: "bond0"
    mode: 1                              # ACTIVE_BACKUP MODE
    slaveInterfaces:
    - "enp1s0f1"
    - "enp2s0f1"

# rate limit for broadcast/multicast traffic on fabric interfaces in bytes per second
fabricBMCastRateLimit: 0
```

- jcnr-cni specific `values.yaml`

  This **values.yaml** file is specific to the jcnr-cni Pod. The jcnr-cni specfic values.yaml file is located under the **Juniper_Cloud_Native_Router_<release-number>/helm_charts/jcnr/charts/jcnr-cni** directory. If you enter any values in this file that conflict with values in the main **values.yaml** file, the values in the main **values.yaml** file take precedence.

```
# Default values for jcnr.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

common:
  registry: svl-artifactory.juniper.net/
  repository: junos-docker-local/warthog/

crpdImage:
  image: crpd
  tag: "22.3R1.8"
  pullPolicy: IfNotPresent

jcnrCNIImage:
  image: jcnr-cni
```

```
  tag: "20220918-fadf886"
  pullPolicy: IfNotPresent

crpdConfigGeneratorImage:
  image: crpdconfig-generator
  tag: "v3"
  pullPolicy: IfNotPresent

busyBox:
  image: busybox
  tag:  "latest"
  pullPolicy: IfNotPresent


#data plane default is dpdk for vrouter case, linux for kernel module
dataplane: dpdk

networkAttachmentDefinitionName: vswitch

crpd_log_path: "/var/log/jcnr/"

# Defines the log severity. Possible options: panic, fatal, error,
# warn or warning, info, debug, trace

log_level: "info"

# parameters for cRPD livenessProbe
livenessProbe:
  initialDelaySeconds: 5
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 3
  successThreshold: 1

# parameters for cRPD startupProbe
startupProbe:
  initialDelaySeconds: 5
  periodSeconds: 10
  timeoutSeconds: 5
  failureThreshold: 3
  successThreshold: 1

crpdConfigs:
```

```
  interface_groups:
    fabricInterface: # TOR side interface, expected all types of traffic
      - bond0:
          interface_mode: trunk # interface mode is always trunk for fabricInterface
          vlan-id-list: [100, 200, 700] # vlan-id-lists
      - enp4s0f0vf0:
          interface_mode: trunk # interface mode is always trunk for fabricInterface
          vlan-id-list: [300, 500, 3001, 3002] # vlan-id-lists
      - enp4s0f0vf1:
          interface_mode: trunk # interface mode is always trunk for fabricInterface
          vlan-id-list: [3003, 3004, 3201-3250, 900] # vlan-id-lists
      - enp4s0f0vf2:
          interface_mode: trunk # interface mode is always trunk for fabricInterface
          vlan-id-list: [3251-3255] # vlan-id-lists
    fabricWorkloadInterface:  # RU side interfaces, expected traffic is only management/
control traffic
      - enp4s0f1vf0:
          interface_mode: access # interface mode is always access for fabricWorkloadInterface
          vlan-id-list: [700] # vlan-id-list must always be a single value for
fabricWorkloadInterface
      - enp4s1f1vf0:
          interface_mode: access # interface mode is always access for fabricWorkloadInterface
          vlan-id-list: [900] # vlan-id-list must always be a single value for
fabricWorkloadInterface

  routing_instances:
    - vswitch:
        instance-type: virtual-switch
```

- nad-dpdk_trunk_vlan_3002.yaml

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: nad-vswitch-bd3002
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "nad-vswitch-bd3002",
    "capabilities":{"ips":true},
    "plugins": [
```

```
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vswitch",
          "instanceType": "virtual-switch",
          "bridgeDomain": "bd3002",
          "bridgeVlanId": "3002",
          "dataplane":"dpdk",
          "mtu": "9000"
        },
        "ipam": {
          "type": "static",
          "capabilities":{"ips":true},
          "addresses":[
            {
              "address":"2001:db8:3002::10.2.0.1/64",
              "gateway":"2001:db83002::10.2.0.254"
            },
            {
              "address":"10.2.0.1/24",
              "gateway":"10.2.0.254"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

- nad-kernel_access_vlan_3001.yaml

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pod1-vswitch-bd3001-1
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "pod1-vswitch-bd3001-1",
    "capabilities":{"ips":true},
    "plugins": [
```

```
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vswitch",
          "instanceType": "virtual-switch",
          "bridgeDomain": "bd3001",
          "bridgeVlanId": "3001",
          "dataplane":"dpdk",
          "mtu": "9000",
          "interfaceType":"veth"
        },
        "ipam": {
          "type": "static",
          "capabilities":{"ips":true},
          "addresses":[
            {
              "address":"2001:db8:3001::10.1.0.1/64",
              "gateway":"2001:db8:3001::10.1.0.254"
            },
            {
              "address":"10.1.0.1/24",
              "gateway":"10.1.0.254"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

- nad-odu-bd3003-sub.yaml

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vswitch-bd3003-sub
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vswitch-bd3003-sub",
    "capabilities":{"ips":true},
```

```
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vswitch",
          "instanceType": "virtual-switch",
          "bridgeDomain": "bd3003",
          "bridgeVlanId": "3003",
          "parentInterface":"net1",
          "interface":"net1.3003",
          "dataplane":"dpdk"
        },
        "ipam": {
          "type": "static",
          "capabilities":{"ips":true},
          "addresses":[
            {
              "address":"10.3.0.1/24",
              "gateway":"10.3.0.254"
            },
            {
              "address":"2001:db8:3003::10.3.0.1/120",
              "gateway":"2001:db8:3003::10.3.0.1"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

- nad-odu-bd3004-sub.yaml

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: vswitch-bd3004-sub
spec:
  config: '{
    "cniVersion":"0.4.0",
    "name": "vswitch-bd3004-sub",
```

```
    "capabilities":{"ips":true},
    "plugins": [
      {
        "type": "jcnr",
        "args": {
          "instanceName": "vswitch",
          "instanceType": "virtual-switch",
          "bridgeDomain": "bd3004",
          "bridgeVlanId": "3004",
          "parentInterface":"net1",
          "interface":"net1.3004",
          "dataplane":"dpdk"

        },
        "ipam": {
          "type": "static",
          "capabilities":{"ips":true},
          "addresses":[
            {
              "address":"30.4.0.1/24",
              "gateway":"30.4.0.254"
            },
            {
              "address":"2001:db8:3004::10.4.0.1/120",
              "gateway":"2001:db8:3004::10.4.0.1"
            }
          ]
        },
        "kubeConfig":"/etc/kubernetes/kubelet.conf"
      }
    ]
  }'
```

- odu-virtio-subinterface.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name:   odu-subinterface-1
  annotations:
    k8s.v1.cni.cncf.io/networks: |
```

```
      [
        {
          "name": "vswitch-bd3003-sub"
        },
        {
          "name": "vswitch-bd3004-sub"
          }
      ]
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - 5d7s39.englab.juniper.net
  containers:
    - name: odu-subinterface
      image: svl-artifactory.juniper.net/junos-docker-local/warthog/pktgen19116:subint
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: false
      resources:
        requests:
          memory: 2Gi
        limits:
          hugepages-1Gi: 2Gi
      env:
        - name: KUBERNETES_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      volumeMounts:
        - name: dpdk
          mountPath: /dpdk
          subPathExpr: $(KUBERNETES_POD_UID)
        - mountPath: /dev/hugepages
          name: hugepage
  volumes:
    - name: dpdk
      hostPath:
```

```
        path: /var/run/jcnr/containers
    - name: hugepage
      emptyDir:
        medium: HugePages
```

- pod-dpdk-trunk-vlan3002.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name:   odu-trunk-1
  annotations:
    k8s.v1.cni.cncf.io/networks: nad-vswitch-bd3002
spec:
  containers:
    - name: odu-trunk
      image: svl-artifactory.juniper.net/junos-docker-local/warthog/pktgen19116:trunk
      imagePullPolicy: IfNotPresent
      securityContext:
        privileged: true
      resources:
        requests:
          memory: 2Gi
        limits:
          hugepages-1Gi: 2Gi
      env:
        - name: KUBERNETES_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      volumeMounts:
        - name: dpdk
          mountPath: /dpdk
          subPathExpr: $(KUBERNETES_POD_UID)
        - mountPath: /dev/hugepages
          name: hugepage
  volumes:
    - name: dpdk
      hostPath:
        path: /var/run/jcnr/containers
    - name: hugepage
```

```
      emptyDir:
        medium: HugePages
```

- pod-kernel-access-vlan-3001.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name:   odu-kenel-pod-bd3001-1
  annotations:
    k8s.v1.cni.cncf.io/networks: pod1-vswitch-bd3001-1
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - 5d8s7.englab.juniper.net
  containers:
    - name: odu-kenel-pod-bd3001-1
      image: vinod-iperf3:latest
      imagePullPolicy: IfNotPresent
      command: ["/bin/bash","-c","sleep infinity"]
      securityContext:
        privileged: false
      env:
        - name: KUBERNETES_POD_UID
          valueFrom:
            fieldRef:
               fieldPath: metadata.uid
      volumeMounts:
        - name: dpdk
          mountPath: /dpdk
          subPathExpr: $(KUBERNETES_POD_UID)
  volumes:
    - name: dpdk
```

```
      hostPath:
        path: /var/run/jcnr/containers
```