Engineering Simplicity

Cloud Native Contrail Networking

Cloud-Native Contrail Networking Feature Guide

Published 2023-09-13

RELEASE 22.3

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

Cloud Native Contrail Networking Cloud-Native Contrail Networking Feature Guide
22.3

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

Table of Contents

About This Guide

This guide provides an understanding of the features and tasks that you can configure for Juniper Cloud-Native Contrail® Networking™ (CN2), Release 22.3. This guide is appropriate for administrators and operators who need to know how to use CN2.

1

CHAPTER

Configure Kubernetes and Contrail

Enable IP Fabric Forwarding and Fabric Source NAT

IN THIS SECTION

This topic shows you how to enable IP fabric forwarding and fabric source NAT in Kubernetes-orchestrated environments using Juniper Networks' Cloud-Native Contrail® Networking™ Release 22.1 or later.

Cloud-Native Contrail Networking supports IP fabric forwarding and fabric source NAT. With IP fabric forwarding, clusters running in the overlay network access the underlay network through the external virtual network. Fabric source NAT enables a gateway device in a fabric to translate the source IP address of data plane node traffic exiting the fabric into a public-side IP address.

You can use IP fabric forwarding and fabric source NAT in cloud-networking environments to provide access to the underlay network. This underlay network access does not add significant network complexity like other underlay network options, such as complex BGP topologies or firewall setups.

The underlay network access enables resources within pods to directly access the Internet or to pull external artifacts from the underlay network.

Overview: IP Fabric Forwarding

Starting in Release 22.1, Cloud-Native Contrail supports IP fabric forwarding in Kubernetes environments.

You enable IP fabric forwarding within virtual networks that have access to the external network. These virtual networks require direct access to the underlay network.

A virtual network that has access to the external network is named the *default-externalnetwork* by default. You can create a customized user-defined external network name, if you choose. When you enable IP fabric forwarding, the path to the underlay network is directly available to clusters running in the overlay network through this external virtual network. This direct connection between the overlay network and the underlay network gives hosts in the overlay network access to the underlay network. Because IP fabric forwarding enables a virtual network to span both the overlay network and the underlay network, data packets traversing the two networks are not encapsulated and de-encapsulated. Packet processing, therefore, is more efficient.

IP fabric forwarding is also extremely useful for load balancing network traffic. A LoadBalancer service automatically detects any external virtual network that has enabled IP fabric forwarding when load-balancing external network traffic.

Overview: Fabric Source NAT

Starting in Release 22.1, Cloud-Native Contrail supports fabric source NAT in Kubernetes environments. Fabric source NAT provides a method for traffic from a data plane node in a Kubernetes environment to directly access the Internet without traversing a separate NAT firewall. You can also use source NAT to pull external artifacts into pods when needed.

Traffic from data plane nodes destined for the Internet must traverse a gateway device. This gateway device is a member device in the fabric that also has at least one interface connected to the public network. When you enable fabric source NAT , the gateway device translates the source IP address of the originating packet from the data plane node into its own public side IP address. This address translation allows traffic from the data plane node to access the Internet.

The IP address translation that source NAT performs also updates the source port in the packet. Multiple data plane nodes can reach the public network through a single gateway public IP address using fabric source NAT.

You need fabric source NAT to translate the IP addresses of traffic exiting the fabric to the Internet. You are not using NAT to translate incoming traffic with this feature.

Example: Configure Fabric Source NAT

Fabric source NAT is disabled by default in user-created virtual networks.

You can enable fabric source NAT manually in any individual virtual network by setting the *fabricsource NAT:* variable in the *VirtualNetwork* object to **true**. You can disable fabric source NAT by setting this value to **false**.

Following is an example of a virtual network object that has enabled fabric source NAT:

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualNetwork
metadata:
  namespace: contrail
  name: virtualnetwork-sample
  annotations:
    core.juniper.net/display-name: Sample Virtual Network
    core.juniper.net/description:
      VirtualNetwork is a collection of end points (interface or ip(s) or MAC(s))
      that can communicate with each other by default. It is a collection of
      subnets whose default gateways are connected by an implicit router
spec:
  ...
  fabricsource NAT: true
```

You can also configure your environment to enable fabric source NAT in any user-created virtual network when the virtual network is created. If you want to enable fabric source NAT in any user-created virtual network upon creation, set the *enablesource NAT* variable in the *ApiServer* resource to **true** when initially deploying your environment.

You must set this configuration in the *ApiServer* resource during initial deployment. You cannot change this setting in your environment after you apply the deployment YAML file. If you want to change the fabric source NAT setting for an individual virtual network after initial deployment, you must change the configuration manually for that network.

Following is a representative YAML file configuration:

```
    apiVersion: configplane.juniper.net/v1alpha1
    kind: ApiServer
    metadata:
      ...
    spec:
      enablesource NAT: true
      common:
        containers:
        ...
```

Fabric source NAT is enabled in any user-created virtual network upon creation when the *enablesource NAT* variable is **true**. You can disable fabric source NAT when user-created virtual networks are created by setting the *enablesource NAT* variable to **false**. Fabric source NAT is disabled by default.

Fabric source NAT automatically selects the IP addresses for translation. You do not need to configure address pools for fabric source NAT in most Cloud-Native Contrail Networking use cases. Address pools are configurable, however, using the *portTranslationPools:* hierarchy within the *GlobalVrouterConfig* resource.

Example: Configure External Networks with IP Fabric Forwarding

IP fabric forwarding is disabled by default.

You can enable IP fabric forwarding in any virtual network by setting the *fabricForwarding:* variable in the *v4SubnetReference:* or *v6SubnetReference:* hierarchies to **true**.

Following is an example of how to enable IP fabric forwarding in an external virtual network that accesses the Internet through an IPv4 gateway:

```
kind: VirtualNetwork
metadata:
  namespace: contrail
  name: external-vn
  labels:
    service.contrail.juniper.net/externalNetworkSelector: default-external
  annotations:
    core.juniper.net/display-name: Sample Virtual Network
    core.juniper.net/description:
      VirtualNetwork is a collection of end points (interface or ip(s) or MAC(s))
      that can communicate with each other by default. It is a collection of
      subnets whose default gateways are connected by an implicit router
spec:
  v4SubnetReference:
    apiVersion: core.contrail.juniper.net/v1alpha1
    kind: Subnet
    namespace: contrail
    name: external-subnet
  fabricForwarding: true
```

You can also enable IP fabric forwarding while creating the external virtual network that has a path to the Internet.

You configure a virtual network's path to an external network through the *Kubemanager* resource in environments using Contrail Networking.

You enable external access for a virtual network by connecting the virtual network to an IPv4 or IPv6 gateway IP subnet address. You enable IP fabric forwarding for the external traffic in the virtual network using the same *Kubemanager* resource.

NOTE: You must configure the external network subnets and this IP fabric forwarding setting during the initial Cloud-Native Contrail deployment. You cannot configure these parameters after the initial deployment YAML file is applied.

The following example shows a YAML file used to configure a *Kubemanager* resource that creates a virtual network with external network access. The virtual network in this example runs with IP fabric forwarding. You would have to commit this YAML file during initial deployment.

```
apiVersion: configplane.juniper.net/v1alpha1
  kind: Kubemanager
  metadata:
    ...
  spec:
    externalNetworkV4Subnet: # Fill V4 Subnet of an external network if any
    externalNetworkV6Subnet: # Fill V6 Subnet of an external network if any
    ipFabricFowardingExtSvc: true
    common:
      containers:
      ...
```

You specify the IPv4 subnet or the IPv6 subnet of the external network using the *externalNetworkV4Subnet* or *externalNetworkV6Subnet:* variable in this YAML file. The subnet address is a public-side IP address that is reachable from the Internet through the gateway device. When you configure a Kubemanager resource using this YAML file, a new virtual network to the specified external network is created. This virtual network is named *default-externalnetwork* in the default namespace for Contrail Networking.

IP fabric forwarding runs in the virtual network with external network access when the *ipFabricFowardingExtSvc* variable is **true**. You can disable IP fabric forwarding for the external subnet by setting the *ipFabricFowardingExtSvc* variable to **false**.

Enable Pods with Multiple Network Interfaces

IN THIS SECTION

Cloud-Native Contrail® Networking™ (CN2) supports multiple network interfaces for a pod within Kubernetes.

Cloud-Native Contrail Networking natively supports multiple network interfaces for a pod.

You can also enable multiple network interfaces in Cloud-Native Contrail Networking using Multus. Multus is a container network interface (CNI) plugin for Kubernetes developed by the Kubernetes Network Plumbing Working Group. Cloud-Native Contrail can interoperate with Multus to provide support for multiple interfaces provided by multiple CNIs in a pod.

This document provides the steps to enable multiple interfaces for a pod in environments using Release 22.1 or later in Kubernetes-orchestrated environments. It includes information about when and how to enable multiple networking interfaces.

Multiple Network Interfaces in Cloud-Native Contrail Benefits

Support for multiple network interfaces is useful or required in many cloud-networking environments. These are a few common examples:

- Pods routinely require a data interface to carry data traffic and a separate interface for management traffic.
- Virtualized network functions (VNFs) typically need three interfaces—a left, a right, and a management interface—to provide network functions. A VNF often can't provide its function with a single network interface.

- Cloud network topologies routinely need to support two or more network interfaces to isolate management networks from tenant networks.

- In customized or high-scale cloud-networking environments, you often must use a cloud-networking product that supports multiple network interfaces to meet a variety of environment-specific requirements.

A pod in a Kubernetes cluster using the default CNI has a single network interface for sending and receiving network traffic. Cloud-Native Contrail Networking provides native support for multiple network interfaces. Cloud-Native Contrail Networking also supports Multus integration, allowing environments using Cloud-Native Contrail for networking to support multiple network interfaces using Multus.

Multiple Network Interfaces in Cloud-Native Contrail Overview

You can enable multiple network interfaces in Cloud-Native Contrail using Multus and without using Multus. Multus is a container network interface (CNI) plugin for Kubernetes that enables support for multiple network interfaces on a pod as well as multihoming between pods. Multus can simultaneously support interfaces from multiple delegate CNIs. This support allows for creating interconnected cloud-networking environments using CNIs from different vendors, including Cloud-Native Contrail. Multus is often called a "meta-plugin" because of this multi-vendor support.

The following two paragraphs describe scenarios that lend themselves to the different ways of enabling multiple network interfaces.

You should enable multiple network interfaces using the native Cloud-Native Contrail Networking support for multiple network interfaces if the following criteria apply;

- You do not want the overhead of enabling and maintaining Multus in your environment.

- You are using Cloud-Native Contrail Networking as your only container networking interface (CNI).

- You do not want to create and maintain Network Attachment Definition (NAD) objects to support multiple network interfaces in your environment.

  You must create a NAD object to enable multiple network interfaces with Multus. You do not have to configure a NAD object to enable multiple network interfaces if you are not using Multus.

  Each NAD object creates a virtual network and a subnet that you have to monitor and maintain.

You should enable multiple network interfaces using Multus if the following criteria apply:

- You are using Cloud-Native Contrail in an environment that is already using Multus. Multus is especially common in environments using Openshift orchestration.

- You need the "meta-plugin" capabilities provided by Multus. You are using Cloud-Native Contrail in an environment where a pod is using multiple interfaces and the multiple interfaces are being managed by Cloud-Native Contrail and other CNIs.
- You need some of the other Multus features in your environment.

Cloud-Native Contrail Integration with Multus Overview

A Contrail vRouter is natively Multus-aware. No Cloud-Native Contrail Networking-specific configuration is required to enable Multus interoperability with Cloud-Native Contrail.

This list summarizes Cloud-Native Contrail support interoperability options with Multus:

- Cloud-Native Contrail is compatible with Multus CNI version 0.3.1.
- Cloud-Native Contrail is supported as a primary CNI with Multus. It is not supported with Multus as the secondary CNI.
- Cloud-Native Contrail is supported as a delegate CNI for Multus. Cloud-Native Contrail should function as the default CNI or as one of the delegate CNIs when it is interoperating in a cluster with Multus.
- Cloud-Native Contrail supports interoperability with Multus when in vRouter kernel mode or DPDK mode.

Multus is a third-party plugin. You enable and configure Multus within Kubernetes entirely outside of Cloud-Native Contrail. To enable Multus, you can apply the multus-daemonset.yml files provided by the Kubernetes Network Plumbing Working Group. The Kubernetes Network Plumbing Working Group is the open-source group that develops Multus.

For detailed information about Multus, see the Multus CNI Usage Guide from the Kubernetes Network Plumbing Working Group.

Create a Network Attachment Definition Object

You do not need to create a *NetworkAttachmentDefinition* (NAD) object to enable multiple interfaces using the native multiple interfaces support in Cloud-Native Contrail Networking. You can skip this section if you are not using Multus to enable multiple network interfaces in your environment. If you are not using NAD objects but need to create a virtual network, see https://www.juniper.net/documentation/us/en/software/cn-cloud-native22/cn-cloud-native-feature-guide/cn-cloud-native-network-feature/topics/concept/Contrail_Network_Policy_Implementation_in_CN2.html.

This section illustrates how to create a NAD object using a YAML file. You configure Cloud-Native Contrail into the NAD object using the *juniper.net/networks* annotation. We provide a representative example of the YAML file that creates the NAD object and a field descriptions table later in this section.

Be sure to include the *juniper.net/networks* annotation when you create the *NetworkAttachmentDefinition* object. If you define the YAML file to create the *NetworkAttachmentDefinition* object without using the *juniper.net/networks* annotation, the *NetworkAttachmentDefinition* object is treated as a third-party object. No Contrail-related objects will be created in the network, including the *VirtualNetwork* object and the *Subnet* object.

You create the *NetworkAttachmentDefinition* object in a Kubernetes environment using the NAD controller. The NAD controller runs in kube-manager and either creates a *VirtualNetwork* object or updates an existing *VirtualNetwork* object when a *NetworkAttachmentDefinition* is successfully created. The NAD controller is enabled by default but you can disable it; see "Disable the Network Attachment Definition Controller" on page 13.

Following is an example of the YAML file used to create a *NetworkAttachmentDefinition* object:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: networkname-1
  namespace: nm1
  annotations:
    juniper.net/networks: '{
      "ipamV4Subnet": "172.16.10.0/24",
      "ipamV6Subnet": "2001:db8::/64",
      "routeTargetList": ["target:23:4561"],
      "importRouteTargetList": ["target:10.2.2.2:561"],
      "exportRouteTargetList": ["target:10.1.1.1:561"],
      "fabricSNAT": true
    }'
spec:
  config: '{
  "cniVersion": "0.3.1",
  "name": "juniper-network",
  "type": "contrail-k8s-cni"
}'
```

The *NetworkAttachmentDefinition Object Fields* table provides usage details for the variables in the *NetworkAttachmentDefinition* object file.

Table 1: NetworkAttachmentDefinition Object Fields

Variable	Usage
ipamV4Subnet	(Optional) Specifies the IPv4 subnet address for the virtual network.
ipamV6Subnet	(Optional) Specifies the IPv6 subnet address for the virtual network.
routeTargetList	(Optional) Provides a list of route targets that are used as both import and export routes.
importRouteTargetList	(Optional) Provides a list of route targets that are used as import routes.
exportRouteTargetList	(Optional) Provides a list of route targets that are used as export routes.
fabricSNAT	(Optional) Specifies if you'd like to toggle connectivity to the underlay network using the port-mapping capabilities provided by the fabric source NAT. Set this parameter to **true** or **false**. It is set to **false** by default.

You should note the following network activities related to the *NetworkAttachmentDefinition* object:

- The network attachment definition controller works in kube-manager and handles processing of all network attachment definition objects.
- You can monitor network attachment definition controller updates in *juniper.net/network-status*.
- IPAM updates are not allowed to the network attachment definition object.

The network attachment definition object creates a virtual network. The **Network Attachment Definition Object Impact on Virtual Networks** table provides an overview of how events related to the network attachment definition object impact virtual networks.

Table 2: Network Attachment Definition Object Impact on Virtual Networks

If	Then
You define a namespace for a network attachment definition object in a single cluster topology	A VirtualNetwork is created in the same namespace as the network attachment definition. This VirtualNetwork will have the same name as the Network Attachment Definition object. The NAD object is named using the **name:** field in the **metadata:** hierarchy.
You define a namespace for a network attachment definition object in a multi-cluster topology	The *VirtualNetwork* namespace is *cluster-name*-**ns**..
A namespace is not defined for a network attachment definition object in a multi-cluster topology	The *VirtualNetwork* namespace is *cluster-name*-**default**.
You delete a network attachment definition resource	The associated *VirtualNetwork* object is also deleted.
You delete a virtual network that was created by the network attachment definition resource	The network attachment definition controller reconciles the issue and recreates the virtual network.

Configure a Pod to Use Multiple Interfaces

You configure multiple interfaces in the pod object. If you are using Multus, you must also configure the network attachment definition (NAD) object as outlined in "Create a Network Attachment Definition Object" on page 9.

In the following example, you create two interfaces for network traffic in the *juniper-pod-1* pod: *tap1* and *tap2*.

```
apiVersion: v1
kind: Pod
metadata:
        name: juniper-pod-1
        namespace: juniper-ns
        annotations:
                k8s.v1.cni.cncf.io/networks: |-
```

```
                          [
                                  {
                                           "name":"juniper-network1",
                                           "namespace":"juniper-ns",
                                           "cni-args":null,
                                           "ips":["172.16.20.42"],
                                           "mac":"de:ad:00:00:be:ef",
                                           "interface":"tap1"
                                  },
                          [
                                  {
                                           "name":"juniper-network2",
                                           "namespace":"juniper-ns",
                                           "cni-args":null,
                                           "ips":["172.16.21.42"],
                                           "mac":"de:ad:00:00:be:ee",
                                           "interface":"tap2"
                                  }
                          ]
```


Disable the Network Attachment Definition Controller

The network attachment definition (NAD) controller is part of the kube-manager object. You enable and disable this controller using the **enableNad:** variable within the YAML file that defines the **kubemanager** object. The network attachment definition controller is enabled by default.

You might want to disable the network attachment definition controller to prevent the application of *NetworkAttachmentDefinion* objects.

In the following example, the network attachment definition controller is disabled:

```
kind: Kubemanager
metadata:
  name: remote-cluster
  namespace: contrail
spec:
  common:
    nodeSelector:
```

```
    node-role.kubernetes.io/master: ""
enableNad: false
```

Encrypt Secret Data at Rest

Juniper Cloud-Native Contrail Networking (CN2) automatically encrypts secret data at rest in your Kubernetes cluster and encrypts any password that you configure. A secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Data at rest encryption is a cybersecurity practice of encrypting stored data to prevent unauthorized access.

See the Kubernetes documentation titled Encrypting Secret Data at Rest for more information.

2

CHAPTER

Advanced Virtual Networking

Enable BGP as a Service

IN THIS SECTION

Cloud-Native Contrail® Networking™ supports BGP as a Service (BGPaaS). This document should be used to enable BGPaaS in environments using Release 22.1 or later.

The BGPaaS feature in Cloud-Native Contrail Networking provides the network support for BGP to operate within a virtual network in cloud networking environments using Cloud-Native Contrail Networking.

Benefits of BGP as a Service in Cloud-Native Contrail

With BGPaaS in Kubernetes environments using Cloud-Native Contrail Networking, you gain the following functionality:

- A BGP protocol service that runs in the virtual network. This BGP service creates BGP neighbor sessions to pods, virtual machines, and other workloads in the virtual network.
- A routing protocol that supports IPv4 neighbors, the IPv4 and IPv6 unicast address family, and IPv6-over-IPv4 next-hop mapping.
- A BGP protocol service that is user-configurable using most well-known BGP configuration parameters.

You can use BGPaaS in any cloud networking environment that needs the functionality provided by a routing protocol. You may find BGPaaS especially useful in the following scenarios:

- If you manage a large cloud networking environment that runs multiple workloads, you may want to use BGPaaS to scale network services.
- If you use tunneling protocols that need network reachability information from a routing protocol to create and maintain tunnels, BGPaaS can help.

Prerequisites

We assume that before you enable BGP as a service:

- You are operating in a working cloud networking environment using Kubernetes orchestration, and Cloud-Native Contrail Networking is operational.
- You have a working knowledge of BGP.

Overview of BGP as a Service in Cloud-Native Contrail Networking

Cloud-Native Contrail Networking provides the networking support for BGPaaS.

You have to find a BGP service to run BGP in your cloud networking environment. This topic shows how to enable networking support for BGPaaS with Cloud-Native Contrail Networking using the BGP service provided by the BIRD Internet Routing Daemon (BIRD). This daemon is available as a built-in development tool on many versions of Unix. You can also download it to your environment using a separate image.

In the examples that follow, you see that the BGP daemon from BIRD runs in a pod when BGPaaS is enabled. That daemon then sends BGP messages over the network using the networking capabilities provided by Cloud-Native Contrail Networking. For additional information on BIRD, see the BIRD Internet Routing Daemon homepage.

When BGPaaS is operational, the BGP daemon runs in a pod and manages BGPaaS. The BGP daemon is directly connected to a Contrail vRouter.

The Contrail vRouter has a connection to at least one control plane node and connects the BIRD daemon to the control plane. A BGP peering session between at least one control node and the BIRD daemon is established through this connection with the Contrail vRouter.

After a peering session is created between the control nodes and the BGP daemon, the BGP daemon can manage BGPaaS and send routes to BGP clients over the control plane. The BGPaaS management tasks include assigning IP addresses to workloads, pods, VMs, or other objects.

Enable BGPaaS in a Pod

To enable BGPaaS, you must create a pod to host the BGP service. You must then associate the pod hosting the BGP service with the virtual networks where BGPaaS will run.

You can use either of two methods of associating a pod hosting the BGP service with a virtual network:

- Virtual Machine Interfaces Selector—The pod running the BGP service is directly associated with the virtual network. The pod hosting the BGP service is discovered automatically after the virtual network association is defined.
- Virtual Machine Interface References—The pod running the BGP service is directly associated with the virtual network by explicitly providing the namespace and the name of the virtual machine interface of the pod hosting the BGP service.

The following sections provide the steps for each association method.

Enable BGPaaS in a Pod Using the Virtual Machine Interfaces Selector

You must create a pod to host the BGP service, and then you can enable BGPaaS with the Virtual Machine Interfaces Selector.

The pod must:

- Include at least one IPv4 interface.
- Include annotations using `core.juniper.net/bgpaas-networks` to specify the associated virtual network names. The value in this annotation must include at least one virtual network name. If you are associating the pod hosting the BIRD daemon with multiple virtual networks, enter the virtual network names as a comma-separated list.

In this example YAML file, a pod is created to host the BGP service. The pod is associated with two virtual networks and BGPaaS is enabled to run on both virtual networks. The **image:** variable in the **containers:** hierarchy points to the BIRD image file that will provide the BGP service in this example.

```
apiVersion: v1
kind: Pod
metadata:
  name: bird-pod-shared-1
  namespace: bgpaas-ns
  annotations:
    k8s.v1.cni.cncf.io/networks: |
        [{
          "name": "bgpaas-vn-1",
```

```
          "namespace": "bgpaas-ns",
          "cni-args": null
          "interface": "eth1"
        },{
          "name": "bgpaas-vn-2",
          "namespace": "bgpaas-ns",
          "cni-args": null
          "interface": "eth2"
        }]
    core.juniper.net/bgpaas-networks: bgpaas-vn-1,bgapss-vn-2
spec:
  containers:
    - name: bird-pod-c
      image: somewhere.juniper.net/cn2/bazel-build/dev/bird-sut:1.0
      command: ["bash","-c","while true; do sleep 60s; done"]
      securityContext:
        privileged: true
```

Enter the **kubectl get vmi -n** ***virtual-network-name*** to confirm that the pod and its associated virtual machine interfaces have been created. You can also enter the **kubectl describe** command to ensure that the virtual machine interfaces exist.

You can confirm the virtual network was created by reviewing the **bgpaasVN=** output in the label section of the **kubectl describe** command.

```
kubectl get vmi -n bgpaas-ns
  CLUSTERNAME                         NAME                 NETWORK              PODNAME
IFCNAME   STATE     AGE
  contrail-k8s-kubemanager-kubernetes   bird-pod-1-abb881a8   bgpaas-vn-1          bird-pod-1
eth1      Success   13s
  contrail-k8s-kubemanager-kubernetes   bird-pod-1-e3f93f05   default-podnetwork   bird-pod-1
eth0      Success   13s

  kubectl describe vmi bird-pod-1-abb881a8 -n bgpaas-ns
  Name:         bird-pod-1-abb881a8
  Namespace:    bgpaas-ns
  Labels:       core.juniper.net/bgpaasVN=bgpaas-vn-1
                namespace=bgpaas-ns
```

You must then create a BGPaaS object to configure BGPaaS. The BGPaaS object references the virtual networks in the **virtualMachineInterfacesSelector:** section.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: BGPAsAService
metadata:
  namespace: bgpaas-ns
  name: bgpaas-test
spec:
  shared: false
  autonomousSystem: 10
  bgpAsAServiceSessionAttributes:
    loopCount: 2
    routeOriginOverride:
    origin: EGP
    addressFamilies:
      family:
        - inet
        - inet6
  virtualMachineInterfacesSelector:
    - matchLabels:
        core.juniper.net/bgpaasVN: bgpaas-vn-1
    - matchLabels:
        core.juniper.net/bgpaasVN: bgpaas-vn-2
```

Enable BGPaaS in a Pod Using Virtual Machine Interface References

You must first create a pod to host the BIRD daemon to enable BGPaaS with Virtual Machine Interface references. The pod must include at least one IPv4 interface.

In the following example, a pod is created in the **bgpaas-ns** namespace. The annotation associates the pod with the **bgpaas-vn-1** virtual network. The **image:** variable in the **containers:** hierarchy points to the BIRD image file that will provide the BGP service in this example.

```
apiVersion: v1
kind: Pod
metadata:
  name: bird-pod-1
  namespace: bgpaas-ns
  annotations:
    k8s.v1.cni.cncf.io/networks: bgpaas-vn-1
```

```
spec:
  containers:
    - name: bird-pod-c
      image: somewhere.juniper.net/cn2/bazel-build/dev/bird-sut:1.0
      command: ["bash","-c","while true; do sleep 60s; done"]
      securityContext:
        privileged: true
```

Confirm that the pod was created after committing the pod object configuration file by entering the **kubectl get vmi -n bgpaas-ns** command.

Note the name of the virtual machine interface for the pod in this command output. You will need to specify the virtual machine interface name later in this procedure when configuring the BGPaaS object.

```
kubectl get vmi -n bgpaas-ns
  CLUSTERNAME                          NAME                 NETWORK              PODNAME
IFCNAME   STATE     AGE
  contrail-k8s-kubemanager-kubernetes  bird-pod-1-abb881a8  bgpaas-vn-1          bird-pod-1
eth1      Success   13s
  contrail-k8s-kubemanager-kubernetes  bird-pod-1-e3f93f05  default-podnetwork   bird-pod-1
eth0      Success   13s
```

The Virtual Machine interface references are defined while creating the BGPaaS object using the **virtualMachineInterfaceReferences:** hierarchy. The **namespace:** is the pod namespace and the **name:** is the virtual machine interface name that you retrieved using the **kubectl get vmi -n bgpaas-ns** command.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: BGPAsAService
metadata:
  namespace: bgpaas-ns
  name: bgpaas-sample
spec:
  shared: false
  autonomousSystem: 100
  bgpAsAServiceSessionAttributes:
    localAutonomousSystem: 10
    loopCount: 2
    routeOriginOverride:
      origin: EGP
    addressFamilies:
      family:
        - inet
```

```
        - inet6
  virtualMachineInterfaceReferences:
    - apiVersion: core.contrail.juniper.net/v1alpha1
      kind: VirtualMachineInterface
      namespace: bgpaas-ns
      name: bird-pod-1-abb881a8
```

Configure the IP Address Allocation Method for BGPasaS

You can configure BGPaaService with one of the following IP address allocation methods:

- automatic IP address allocation—The BGP service assigns IP addresses.
- user-specified IP address allocation—You assign the IP address.

You configure the IP address allocation method in the `Subnet` object.

Automatic IP address allocation is enabled by default. If you enable BGPaaS without manually disabling automatic IP address allocation, BGPaaS uses automatic IP address allocation.

You disable automatic IP address allocation by setting the *disableBGPaaSIPAutoAllocation:* variable in the `Subnet` object to *true*. If the *disableBGPaaSIPAutoAllocation:* variable is not present in the **Subnet** object file, automatic IP address allocation is enabled.

In the following configuration sample, automatic IP address allocation is enabled because the *disableBGPaaSIPAutoAllocation:* variable isn't present in the **Subnet** object configuration file.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: Subnet
metadata:
  namespace: bgpaas-ns
  name: bgpaas-subnet-1
spec:
  cidr: "172.20.10.0/24"
```

In this configuration sample, automatic IP address allocation is enabled because the *disableBGPaaSIPAutoAllocation:* variable is set to **false**.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: Subnet
metadata:
```

```
  namespace: bgpaas-ns
  name: bgpaas-subnet-2
spec:
  cidr: "172.20.20.0/24"
 disableBGPaaSIPAutoAllocation: false
```

To enable user-specified IP address allocation, set the *disableBGPaaSIPAutoAllocation:* variable to `true`. When user-specified IP address allocation is enabled, you must also configure the BGP addresses that BGPaaS can assign to endpoints within the subnet. You must set a primary IP address using the *bgpaasPrimaryIP:* variable. You can also set an optional secondary IP address, which you can see in this example with the *bgpaasSecondaryIP:* variable.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: Subnet
metadata:
  namespace: bgpaas-ns
  name: bgpaas-subnet-2
spec:
  cidr: "172.20.20.0/24"
 disableBGPaaSIPAutoAllocation: true
  bgpaasPrimaryIP: 172.20.20.3
  bgpaasSecondaryIP: 172.20.20.4
```

Configure the BGPasaService Object

You enable BGPaaS in a cluster by creating a *BGPAsAService* object.

Create the *BGPAsAService* object by creating a YAML file that uses `BGPAsAService` in the **kind:** field:

```
  apiVersion: core.contrail.juniper.net/v1alpha1
  kind: BGPAsAService
  metadata:
    namespace: bgpaas-ns
    name: bgpaas-test
  spec:
    shared: false
    autonomousSystem: 10
    bgpAsAServiceSessionAttributes:
      loopCount: 2
```

```
    routeOriginOverride:
    origin: EGP
    addressFamilies:
      family:
        - inet
        - inet6
  virtualMachineInterfacesSelector:
    - matchLabels:
        core.juniper.net/bgpaasVN: bgpaas-vn-1
    - matchLabels:
        core.juniper.net/bgpaasVN: bgpaas-vn-2
```

Table 3: Spec Field Variables for BGPaaS

This table provides a description of each Spec field variable in the BGPaaS object file.

Field Variable	Description
shared:	Specifies whether a common BGP router object can be shared with multiple virtual machine interfaces in the same virtual network. When this field is set to **true**, one BGP client router can be shared with multiple virtual machine interfaces in the same virtual network. This setting limits the total number of BGP client routers that have to be created in a virtual network for a VMI. When this field is set to **false**, one BGP client router is created for each virtual machine interface.
autonomousSystem:	Specifies the global Autonomous System number for the BGP instance. The Autonomous System number can be any whole number between 1 and 4294967295.
bgpAsAServiceSessionAttributes:	Defines the BGP session attributes for BGPaaS. See Table 4 on page 25.

Table 3: Spec Field Variables for BGPaaS *(Continued)*

Field Variable	Description
VirtualMachineInterfaceReferences:	Defines the virtual machine interface parameters to associate with BGPaaS when using virtual machine interface references. See Table 5 on page 26.
virtualMachineInterfacesSelector:	Defines the virtual networks where BGPaaS runs when using the virtual machine interfaces selector. See Table 6 on page 26.

Table 4: BgpAsAServiceSessionAttributes Fields for BGPaaS

The `bgpAsAServiceSessionAttributes:` in the `spec:` hierarchy are used in all BGPaaS setups. The `bgpAsAServiceSessionAttributes:` hierarchy includes these fields:

Field	Description
localAutonomousSystem:	Specifies the local Autonomous System number for BGP.
LoopCount:	Specifies the number of times that the same ASN can be seen in a route update before the route is discarded. The **LoopCount:** can be any whole number up to 16.
routeOriginOverride:	Overrides the original setting and sets the origin attribute to **Incomplete** when forwarding routes. If you set this field to **false**, routes are advertised into BGP based on the origin setting. The origin is either IGP or EGP and is set using the **origin:** field in this file. If you set this field to **true**, the origin is set to **Incomplete** for advertised routes. Use this field if you want to change how BGP networks prioritize routes received from the BGP service. By default, BGP networks prioritize routes based on origin, and routes with an **Incomplete** origin receive lower priority than routes received from IGP or EGP.

Table 4: BgpAsAServiceSessionAttributes Fields for BGPaaS *(Continued)*

Field	Description
Origin:	Specifies if BGP operates as an interior gateway protocol (**igp**) or exterior gateway protocol (**egp**). The default route origin is **igp**.
AddressFamilies:	Specifies the address family. You can specify the family as **inet** for IPv4 or **inet6** for IPv6. Both address families can be specified simultaneously.

Table 5: virtualMachineInterfaceReferences: in BGPaaS

The `virtualMachineInterfaceReferences:` in the `spec:` hierarchy include the following fields:

Field	Description
apiVersion:	Specifies the API version for the virtual machine interface reference.
kind:	Always set this field to **VirtualMachineInterface**.
namespace:	Specifies the namespace associated with the virtual machine interface reference. You define this namespace while creating the **Pod** object. See .
name:	Specifies the name of the pod associated with the virtual machine interface reference. You can retrieve the pod name by entering the **kubectl get vmi -n bgpaas-ns** command. See "Enable BGPaaS in a Pod Using Virtual Machine Interface References" on page 20.

Table 6: The virtualMachineInterfacesSelector: Fields in BGPaaS

The `virtualMachineInterfacesSelector:` in the `spec:` hierarchy includes the following fields:

Field	Description
matchLabels:	Define the match labels for the Virtual Machine Interfaces selector. The match labels in this context are always used to reference the virtual networks where the Virtual Machine interfaces selector is running. The match label values in this section are always entered as **core.juniper.net/bgpaasVN:***virtual-network-name*. See "Enable BGPaaS in a Pod Using the Virtual Machine Interfaces Selector" on page 18.

Validate the BGP as a Service Configuration

You should confirm that the BGPaaS object is successfully running after committing the BGPAsAService object file.

Enter the **kubectl get BGPAsAService** command after creating the BGPAsAService object to confirm the object state. The object is successfully created when the **State** field indicates **Success**.

```
kubectl get BGPAsAService -n bgpaas-ns
   NAME            AS    IPADDRESS   SHARED   STATE     AGE
   bgpaas-sample   100               false    Success   33s
```

You should also ensure the BGPaaS server and the BGPaaS client are created and are in the **Success** state.

Enter the **kubectl get BGPRouter** command to confirm the presence and operational state of the BGPaaS servers and clients.

```
kubectl get BGPRouter -n bgpaas-ns
   NAME                                         TYPE            IDENTIFIER    STATE     AGE
   bgpaas-ns-bgpaas-vn-1-bgpaas-server          bgpaas-server                 Success   2m57s
   bgpaas-ns-bgpaas-vn-1-bird-pod-1-abb881a8    bgpaas-client   172.20.10.2   Success   2m57s
```

Configure BGP in Pod

You must also configure the networking parameters for the BGP service running in the pod. The configuration for each individual BGP service is unique. Documenting the required networking configuration parameters is beyond the scope of this document.

In this example, the BGP network configuration is configured using BIRD.

You configure BGP using the BIRD CLI in this example. The parameters of the BGP configuration that need to match the BGPaaS objects defined in Cloud-Native Contrail Networking are noted. Although not shown in this example, you should know that the default location to access the BIRD configuration file in most deployments is /etc/bird.conf or /etc/bird/bird.conf.

```
  # Change the router id to your BIRD router ID. It's a world-wide unique identification
  # of your router, usually one of router's IPv4 addresses.
  router id 172.20.10.2;

  protocol direct {
        interface "eth1*";  -> interface on which BGPAsAService needs to be configured
  }

  protocol bgp bgp1_1 {
        import all;
        export all;
        local as 10;                    -> AS configured in BGPAsAService
        neighbor 172.20.10.3 as 64512;   -> neighbor for primary BGP session, use
BGPaaSPrimaryIP from subnet
        neighbor 172.20.10.3 as 64512;    -> neighbor for secondary BGP session, use
BGPaaSSecondaryIP from subnet
```

You can also verify that the BGP protocol is running from your BGP service.

In this example from BIRD, the **show protocol** command is entered to verify that the BGP protocol is established.

```
birdc show protocol bgp1_1
   BIRD 1.6.8 ready.
   name     proto    table    state  since       info
   bgp1_1   BGP      master   up     10:31:27    Established
```

Create an Isolated Namespace

SUMMARY

This topic shows you how to create an isolated namespace in Juniper Cloud-Native Contrail® Networking™ (CN2). Juniper Networks supports isolated namespaces using Contrail Networking Release 22.1 or later in a Kubernetes-orchestrated environment.

IN THIS SECTION

Namespace Overview

NOTE: In this document, we use the term "isolated" and "non-isolated" in the context of Contrail networking only.

Non-Isolated Namespaces

Namespaces, also called non-isolated namespaces, provide a mechanism for isolating a group of resources within a single cluster. By default, namespaces are not isolated.

Non-isolated namespaces are intended for use in environments with many users spread across multiple teams, or projects. Non-isolated namespaces enable each team to exist in its own virtual cluster without team members affecting each other's work. Let's say that you created all your resources in the default namespace that Kubernetes provides. If you have a complex application with multiple deployments, the default namespace can be hard to maintain. An easier way to manage this deployment is to group all your resources into different namespaces within the cluster. For example, the cluster can contain separate namespaces, such as a database namespace or a monitoring database. Names of resources must be unique within a namespace, but you can repeat resource names across namespaces.

Pods in a non-isolated namespace exhibit the following network behavior:

- They can communicate with other pods in the cluster without using NAT.
- They share the same `default-podnetwork` and `default-servicenetwork`.

Isolated Namespaces.

An isolated namespace enables you to run customer-specific applications that you want to keep private. You can create an isolated namespace to isolate a pod from other pods, without explicitly configuring a network policy.

Isolated namespaces are similar to non-isolated namespaces, except that each isolated namespace has its own pod network and service network. This means that pods in isolated namespaces cannot reach pods or services in other isolated or non-isolated namespaces.

Pods in isolated namespaces can communicate only with pods in the same namespace. The only exception is when a pod in an isolated namespace needs access to a Kubernetes service, such as Core DNS. In this case, the pod uses the cluster's `default-servicenetwork` to access the services.

Pods in an isolated namespace exhibit the following network behavior:

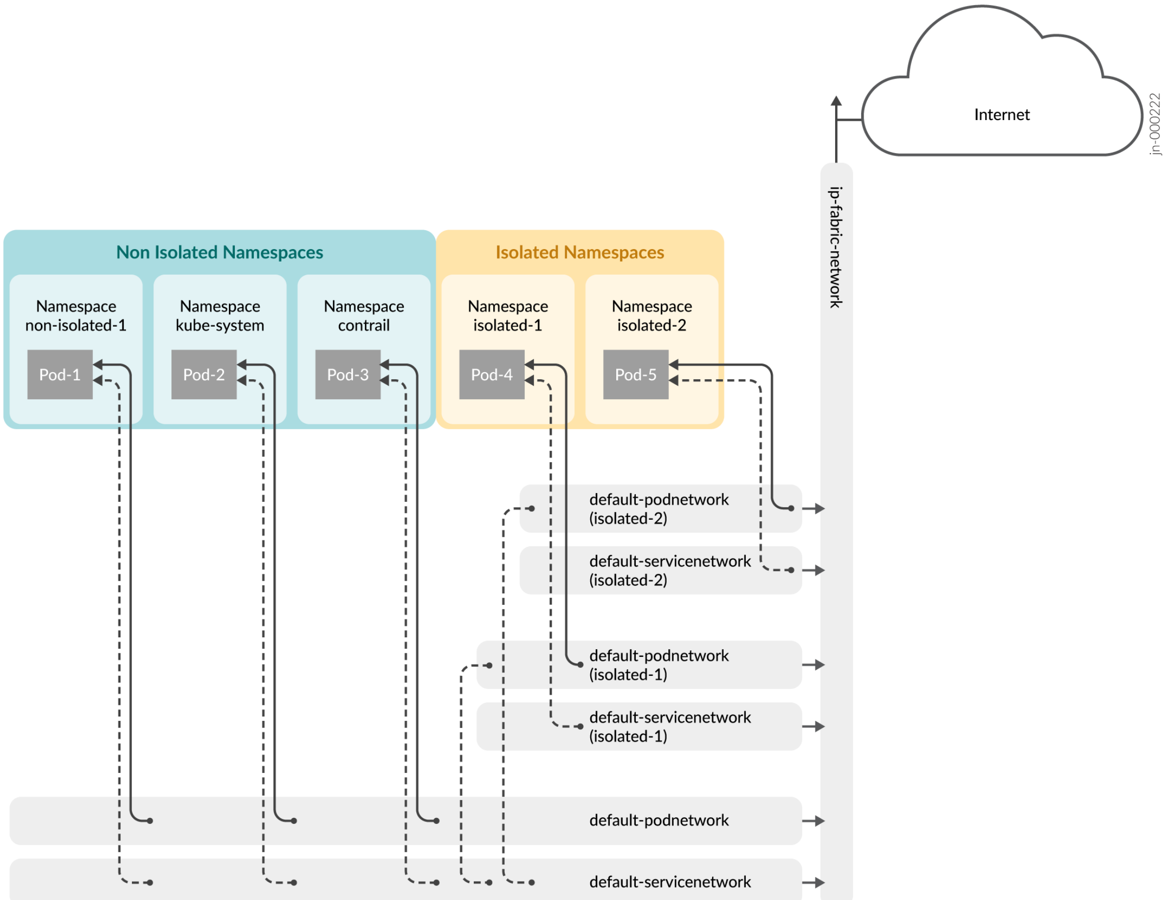
- They can communicate only with pods in the same namespace.
- They can reach services in non-isolated namespaces.
- Their IP addresses and service IP addresses are allocated from the same subnet as the cluster's pod and service subnet.
- They can access the underlay network, or IP fabric network, through IP fabric forwarding and fabric source NAT.

> **NOTE**: You cannot convert a non-isolated namespace to an isolated namespace, or vice versa.

Example: Isolated Namespace Configuration

This sample configuration demonstrates an isolated namespace configuration in Cloud-Native Contrail Networking (CN2).

Figure 1: Isolated Namespace Configuration



In the above isolated namespace configuration:

- Pod-1 (non-isolated-1) is in a non-isolated namespace created by the user.
- Pod-2 (kube-system) and Pod-3 (contrail) are in non-isolated namespaces created by the controller.
- Pod-4 (isolated-1) and Pod-5 (isolated-2) are in isolated namespaces created by the user.
- The interfaces for Pod-1, Pod-2, and Pod-3 are created from the cluster's `default-podnetwork` and `default-servicenetwork`.
- The interfaces for Pod-4 and Pod-5 are created on the `default-podnetwork` and `default-servicenetwork` in their own isolated namespaces. Both Pod-4 and Pod-5 interfaces share the same subnet as the cluster's `default-podnetwork` and `default-servicenetwork`.

- Pods in isolated namespaces cannot communicate with pods in non-isolated namespaces. In this example, Pod-4 and Pod-5 in isolated namespaces cannot communicate with Pod-1, Pod-2, or Pod-3 in non-isolated namespaces.
- Pods in isolated namespaces (Pod-4 and Pod 5) can access any service through the cluster's `default-servicenetwork`.
- Pods in all namespaces (non-isolated and isolated) can connect to the fabric through the cluster's `ip-fabric-network`.

Notes

- Isolated namespaces affect only the pod's default interface. This is because the default interface of pods in an isolated namespace are created on the `default-podnetwork` of the isolated namespace. However, interfaces from user-defined `VirtualNetworks` behave the same way in both isolated and non-isolated namespaces.
- You can create network policies on isolated namespaces to adjust the isolation of pods. The network policy behaves the same for both isolated and non-isolated namespaces.
- Two or more isolated namespaces can be interconnected through the `VirtualNetworkRouter` (VNR).

Here is an example of a `VirtualNetworkRouter` (VNR) configuration used to interconnect the `default-podnetworks` of two isolated namespaces (`ns-isolated-1` and `ns-isolated-2`). In this configuration, the `VirtualNetworkRouter` connects to ns-isolated-1 and ns-isolated-2. This means that pods in these isolated namespaces can communicate with each other.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualNetworkRouter
metadata:
  namespace: ns-isolated-1
  name: vnr-1
  annotations:
    core.juniper.net/display-name: vnr-1
  labels:
      vnr: vnr-1
spec:
  type: mesh
  virtualNetworkSelector:
    matchExpressions:
    - key: core.juniper.net/virtualnetwork
      operator: In
      values:
      - isolated-namespace-pod-virtualnetwork
```

```
  import:
    virtualNetworkRouters:
      - virtualNetworkRouterSelector:
          matchLabels:
            vnr: vnr-2
        namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: ns-isolated-2
---
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualNetworkRouter
metadata:
  namespace: ns-isolated-2
  name: vnr-2
  annotations:
    core.juniper.net/display-name: vnr-2
  labels:
      vnr: vnr-2
spec:
  type: mesh
  virtualNetworkSelector:
    matchExpressions:
    - key: core.juniper.net/virtualnetwork
      operator: In
      values:
      - isolated-namespace-pod-virtualnetwork
  import:
    virtualNetworkRouters:
      - virtualNetworkRouterSelector:
          matchLabels:
            vnr: vnr-1
        namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: ns-isolated-1
```

Isolated Namespace Objects

This table describes the namespace objects (API resources) that the controller creates when you create an isolated namespace.

Table 7: Isolated Namespace Objects

Isolated Namespace Object	Description
`default-podnetwork (VirtualNetwork)`	The default interfaces of pods in an isolated namespace are created in this `default-podnetwork` instead of the cluster's default-network.
`default-servicenetwork (VirtualNetwork)`	The cluster IP of services in isolated namespaces are created in this `default-servicenetwork` instead of the cluster's `default-servicenetwork`.
`IsolatedNamespacePodServiceNetwork (VirtualNetworkRouter)`	This object establishes connectivity between the isolated namespace's `default-podnetwork` and `default-servicenetwork`.
`IsolatedNamespaceIPFabricNetwork (VirtualNetworkRouter)`	This object establishes connectivity between the isolated namespace's `default-podnetwork` and `default-servicenetwork`to the cluster's `ip-fabricnetwork`.
`IsolatedNamespacePodToDefaultService (VirtualNetworkRouter)`	This object establishes connectivity between the isolated namespace's `default-podnetwork` to the cluster's `default-servicenetwork`.

Create an Isolated Namespace

Follow these steps to create an isolated namespace:

1. Create a YAML file called `ns-isolated.yml`.
2. Add the label `core.juniper.net/isolated-namespace` to the namespace metadata and set the variable to `"true"`.

```
apiVersion: v1
kind: Namespace
metadata:
   name: ns-isolated
   labels:
     core.juniper.net/isolated-namespace: "true"
```

3. Issue the `kubectl apply` command to apply the configuration.

```
kubectl apply -f ns.yaml
```

4. To verify your configuration, issue the `kubectl get ns ns-isolated -o yaml` command.

```
apiVersion: v1
kind: Namespace
metadata: `\
  annotations:
    core.juniper.net/forwarding-mode: "false"
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Namespace","metadata":{"annotations":{"core.juniper.net/
forwarding-mode":"false"},"labels":{"core.juniper.net/isolated-namespace":"true"},"name":"ns-
isolated"}}
  creationTimestamp: "2021-10-04T21:47:40Z"
  finalizers:
  - finalizers.core.juniper.net/isns-virtualnetworks-delete
  - finalizers.core.juniper.net/isns-virtualnetworkrouters-delete
  labels:
    core.juniper.net/isolated-namespace: "true"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    …
    …
    …
  name: ns-isolated
  resourceVersion: "4183"
  uid: d25d2b71-2051-4ac5-a738-e9b344235818
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

Success! You created an isolated namespace.

Optional Configuration: IP Fabric Forwarding and Fabric Source NAT

Optionally, you can enable IP fabric forwarding and fabric source NAT on an isolated namespace.

IP fabric forwarding enables virtual networks to be created as part of the underlay network and eliminates the need for encapsulation and de-encapsulation of data. Fabric source NAT allows pods in the overlay to reach the Internet without floating IPs or a logical system.

When you create an isolated namespace, two virtual networks are created, a `default-podnetwork` and a `default-servicenetwork`. By default, IP fabric forwarding and fabric source NAT in these two virtual networks are disabled. You enable IP fabric forwarding or fabric source NAT in the virtual networks by adding “forwarding-mode” annotations for each feature in your isolated namespace YAML file.

Here is an example of the `default-podnetwork` for an isolated namespace with forwarding-mode set to `fabricSNAT`.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualNetwork
metadata:
  annotations:
    core.juniper.net/description: Default Pod Network for IsolatedNamespace (ns-isolated)
    core.juniper.net/display-name: default-podnetwork
    ...
spec:
    ...
    fabricSNAT: true
    ...
```

Enable IP Fabric Forwarding

Follow these steps to enable IP fabric forwarding on an isolated namespace:

1. Add the annotation `core.juniper.net/forwarding-mode: "ip-fabric"` to the namespace metadata.
2. Set the label for the isolated namespace to `"true"`.

```
apiVersion: v1
kind: Namespace
metadata:
 name: ns-isolated
```

```
 annotations:
    core.juniper.net/forwarding-mode: "ip-fabric"
    labels:
      "core.juniper.net/isolated-namespace": "true"
```

3. Issue the `kubectl apply` command to enable IP fabric forwarding.

```
kubectl apply -f ns-isolated.yaml
```

4. Verify your configuration.

```
get vn -n ns-isolated default-podnetwork -o yaml
```

```
spec:
  fabricForwarding: true
  fabricSNAT: false
  fqName:
  - default-domain
  - ns-isolated
  - default-podnetwork
  providerNetworkReference:
    apiVersion: core.contrail.juniper.net/v1alpha1
    fqName:
    - default-domain
    - contrail
    - ip-fabric
    kind: VirtualNetwork
    name: ip-fabric
    namespace: contrail
    resourceVersion: "5629"
    uid: bdb0ae55-d5e5-49b2-803d-d93eea206df0
  v4SubnetReference:
    apiVersion: core.contrail.juniper.net/v1alpha1
    fqName:
    - default-domain
    - contrail-k8s-kubemanager-mycluster-contrail
    - default-podnetwork-pod-v4-subnet
    kind: Subnet
    name: default-podnetwork-pod-v4-subnet
    namespace: contrail-k8s-kubemanager-mycluster-contrail
```

```
    resourceVersion: "4999"
    uid: fc9b9471-3b3e-4a57-80ac-5b9ed806fe94
  virtualNetworkProperties:
    forwardingMode: l3
    rpf: enable
status:
  observation: ""
  state: Success
  virtualNetworkNetworkId: 5
```

Success! You enabled IP fabric forwarding on the isolated namespace.

Enable Fabric Source NAT

NOTE: You can enable fabric source NAT only on the `default-podnetwork`.

Follow these steps to enable fabric source NAT on an isolated namespace:

1. Add the annotation `core.juniper.net/forwarding-mode: "fabric-snat"` to the namespace metadata.
2. Set the label for the isolated namespace to `"true"`.

```
apiVersion: v1
kind: Namespace
metadata:
 name: ns-isolated
 annotations:
    core.juniper.net/forwarding-mode: "fabric-snat"
    labels:
      "core.juniper.net/isolated-namespace": "true"
```

3. Issue the `kubectl apply` command to enable fabric source NAT.

```
kubectl apply -f ns-isolated.yaml
```

4. Verify your configuration.

```
kubectl get vn -n <isolated-namespace-name> default-podnetwork
```

Success! You enabled fabric source NAT on the isolated namespace.

```
spec:
  fabricSNAT: true
  fqName:
  - default-domain
  - ns-isolated-snat
  - default-podnetwork
  v4SubnetReference:
    apiVersion: core.contrail.juniper.net/v1alpha1
    fqName:
    - default-domain
    - contrail-k8s-kubemanager-mycluster-contrail
    - default-podnetwork-pod-v4-subnet
    kind: Subnet
    name: default-podnetwork-pod-v4-subnet
    namespace: contrail-k8s-kubemanager-mycluster-contrail
    resourceVersion: "4999"
    uid: fc9b9471-3b3e-4a57-80ac-5b9ed806fe94
  virtualNetworkProperties:
    forwardingMode: l3
    rpf: enable
status:
  observation: ""
  state: Success
  virtualNetworkNetworkId: 7
```

SEE ALSO

Enable IP Fabric Forwarding and Fabric Source NAT | 2

Configure Allowed Address Pairs

Juniper Networks supports Allowed Address Pairs (AAPs) using Cloud-Native Contrail® Networking™ Release 22.1 or later in a Kubernetes-orchestrated environment.

Allowed address pairs in Contrail Networking enables you to add IP/MAC (CIDR) addresses to the guest interface (`VirtualMachineInterface`) by using a secondary IP address.

When you create a pod in a cluster, each pod automatically obtains its IP address from the virtual machine interface. If your pod is *not* on the same virtual network, you can add an AAP to allow traffic to flow through the port regardless of the subnet. For example, let's say that your pod's IP address is 192.168.2.0. If you define an AAP with subnet 192.168.2.0/24, the AAP allows the pods to communicate with the guest interface. The vRouter forwards the traffic and advertises reachability to the pod.

To configure an AAP, insert the following attribute into your pod YAML file, as shown in the code block that follows:

```
kind: Pod
metadata:
  name: my-pod
  namespace: my-namespace
  annotations:
    k8s.v1.cni.cncf.op/networks: |
      [
        {
          "name": "net-a",
          "cni-args": {
              "net.juniper.contrail.allowedAddressPairs": [{
                "ip": 192.168.2.0/24
                "mac": "02:3f:66:ad:00:e9",
                "addressMode": "active-active"
              }],}
          ...
        },
        {
          "name": "net-b",
          ...
        },
```

The `AllowedAddressPairs` attribute contains a list of allowed address pair definitions, as described in the following table.

Table 8: Allowed Address Pair Definitions

Definition	Description
ip	Specify the external pod IP address through which you want to allow traffic to pass.
mac	(Optional) Specify the MAC address of the external pod.
addressMode	(Optional) Specify a high availability (HA) mode. Choose from *active/active* or *active/standby*. Active/ standby is the default. The addressMode default value is an empty string. *Active/standby* is used for VRRP addresses. *Active/ active* is used for ECMP.

In Kubemanager, the `PodController` watching for Pod events, reads the interface definitions for each new AAP. The controller then generates an `AllowedAddressPair` and adds it to the list of interfaces in the `VirtualMachineInterface`.

Alternative Configuration

Alternatively, you can configure AAP interfaces directly from the `VirtualMachineInterface`.To apply an AAP this configuration, run the following command from the kubectl command-line tool:

```
kubectl patch --namespace project-kubemanager VirtualMachineInterface $VMINAME -p "$(cat ./
aap.yaml)"
```

The preceding command updates the existing `VirtualMachineInterface` with the AAP configuration, as shown in the following code block:

```
spec:
  allowedAddressPairs:
    allowedAddressPair:
      - ip:
          ipPrefix: 192.0.2.0
          ipPrefixLen: 24
```

Enable Packet-Based Forwarding on Virtual Interfaces

IN THIS SECTION

Juniper Networks supports packet-based forwarding on virtual interfaces using Cloud-Native Contrail® Networking™ Release 22.1 or later in a Kubernetes-orchestrated environment.

Overview

By default, Contrail compute nodes use flow mode for packet forwarding on a virtual interface. This means that every vRouter has a flow table to keep track of all flows that passes through it. In flow mode, the virtual interface processes all traffic by analyzing the state or session of traffic. However, there might be instances when you want to switch from flow mode to packet mode. Specifically, to achieve higher traffic forwarding performance, or to get around certain limitations of flow mode.

In packet mode, the virtual interface processes the traffic on a per-packet basis and ignores all flow information. The main advantage of this mode is that the processing type is stateless. Stateless mode means that the virtual interface does not keep track of session information or go through traffic analysis to determine how a session is established.

> **NOTE**: Features that require a network policy (such as ACLs, security groups, floating IP's) are unable to work in packet mode.

Configure Packet Mode on a Virtual Interface

Follow these steps to enable packet mode on a virtual interface.

1. Verify that you are running flow mode. Flow mode is the default forwarding mode.

Generate some traffic by pinging another pod in the same network. In this example, the pod's IP address is 25.26.27.2.

```
root@pod-vn-1:/# ping -q -c5 25.26.27.2
PING 25.26.27.2 (25.26.27.2) 56(84) bytes of data.

--- 25.26.27.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4057ms
rtt min/avg/max/mdev = 0.059/1.721/7.620/2.955 ms
```

2. Use the flow command-line tool to check for flows. The following example indicates that the `virtualMachineInterface` is in flow mode.

```
root@minikube:/# flow -l --match 25.26.27.3
...
...
Listing flows matching ([25.26.27.3]:*)

    Index                Source:Port/Destination:Port                      Proto(V)
-----------------------------------------------------------------------------------
   159692<=>400664       25.26.27.2:28                                       1 (3)
                         25.26.27.3:0
(Gen: 1, K(nh):39, Action:F, Flags:, QOS:-1, S(nh):39,  Stats:5/490,  SPort 64222,
 TTL 0, UnderlayEcmpIdx:0, Sinfo 7.0.0.0)

   400664<=>159692       25.26.27.3:28                                       1 (3)
                         25.26.27.2:0
(Gen: 1, K(nh):33, Action:F, Flags:, QOS:-1, S(nh):33,  Stats:5/490,  SPort 56567,
 TTL 0, UnderlayEcmpIdx:0, Sinfo 5.0.0.0)
```

3. Enable packet mode on the `virtualMachineInterface`.

   Create a patch file named `packet-mode-patch.yaml` and set the VMI policy to `true`.

```
spec:
  virtualMachineInterfaceDisablePolicy:true
```

4. Apply the patch.

```
[user@machine:~]$ kubectl -n vmi-disablepolicy patch vmi pod-vn-1-7d622c4d --patch "$(cat
packet-mode-patch.yaml)"
virtualmachineinterface.core.contrail.juniper.net/pod-vn-1-7d622c4d patched
```

5. After you apply the patch, flow mode switches to packet mode.

```
[user@machine:~]$ kubectl -n vmi-disablepolicy get vmi pod-vn-1-7d622c4d -oyaml |
yq .spec.virtualMachineInterfaceDisablePolicy
true
```

6. Verify that packet mode is active.

   Generate traffic by pinging another pod in the same network that you pinged in Step 1.

```
root@pod-vn-1:/# ping -q -c5 25.26.27.2
PING 25.26.27.2 (25.26.27.2) 56(84) bytes of data.

--- 25.26.27.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4105ms
rtt min/avg/max/mdev = 0.051/2.725/13.388/5.331 ms
```

7. Use the flow command-line tool to check for flows.

```
root@minikube:/# flow -l --match 25.26.27.3
...
...
Listing flows matching ([25.26.27.3]:*)

    Index                Source:Port/Destination:Port                      Proto(V)
```

   Success! No flows exist which indicates you are in packet mode.

Configure Reverse Path Forwarding on Virtual Interfaces

IN THIS SECTION

Juniper Networks supports reverse path forwarding (RPF) on virtual interfaces using Cloud-Native Contrail® Networking™ Release 22.1 or later in a Kubernetes-orchestrated environment.

Overview

Unicast reverse-path-forwarding (RPF) verifies that a packet is sent from a valid source address by performing an RPF check. RPF check is a validation tool that uses the IP routing table to verify whether the source IP address of an incoming packet is arriving from a valid path. RPF helps reduce forwarding of IP packets that might be spoofing an IP address.

When a packet arrives on an interface, RPF performs a forwarding table lookup on the packet's source IP address and checks the incoming interface. The incoming interface must match the interface on which the packet arrived. If the interface does not match, the vRouter drops the packet. If the packet is from a valid path, the vRouter forwards the packet to the destination address.

You can enable or disable source RPF on a per-virtual network basis. By default, RPF is disabled.

- RPF enable

  Whenever a packet reaches the interface, RPF performs a check on the packet's source IP address. All packets are dropped if the route is not learned by the vRouter. Only packets received from the MAC/IP address allocated to the workload are permitted on an interface.

- RPF disable

  Packets from any source are accepted on the interface. A forwarding table lookup is not performed on the incoming packet source IP address.

Enable RPF on a Virtual Interface

Here is an example of a Namespace YAML file you use to configure RPF on a virtual interface. To enable RPF, set the `rpf` variable under `virtualNetworkProperties` to `enable`.

```
apiVersion: v1
kind: Namespace
metadata:
  name: rpf-ns
---
apiVersion: core.contrail.juniper.net/v1alpha1
kind: Subnet
metadata:
  namespace: rpf-ns
  name: rpf-subnet-1
  annotations:
    core.juniper.net/display-name: Sample Subnet
    core.juniper.net/description:
      Subnet represents a block of IP addresses and its configuration.
      IPAM allocates and releases IP address from that block on demand.
      It can be used by different VirtualNetwork in the mean time.
spec:
  cidr: "172.20.10.0/24"
---
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualNetwork
metadata:
  namespace: rpf-ns
  name: rpf-vn-1
  annotations:
    core.juniper.net/display-name: Sample Virtual Network
    core.juniper.net/description:
      VirtualNetwork is a collection of end points (interface or ip(s) or MAC(s))
      that can communicate with each other by default. It is a collection of
      subnets whose default gateways are connected by an implicit router
spec:
  v4SubnetReference:
    apiVersion: core.contrail.juniper.net/v1alpha1
    kind: Subnet
    namespace: rpf-ns
    name: rpf-subnet-1
```

```
  fabricSNAT: true
  virtualNetworkProperties:
    rpf: enable
```

Health Check

SUMMARY

In Cloud-Native Contrail Networking (CN2) Release 22.3, a new health check custom resource object is introduced that associates the virtual machine interface (VMI) to the pod creation and update workflow. The health check resource is a namespace-scoped resource.

IN THIS SECTION

Health Check Overview

The Contrail vRouter agent provides the health check functionality. You can associate a ping or HTTP health check to an interface. If the health check fails, based on the timers and intervals configured in the health check object, the interface is set as administratively down and associated routes are withdrawn. Health check traffic continues to be transmitted in an administratively down state to allow for an interface to recover.

Create a Health Check Object

Use this procedure to create a health check object.

1. In the deployment manifests from the Contrail Networking download page, use the `hc.yaml` file (shown below) for the YAML definition for health check objects. The same folder also includes the `hc_pod.yaml` which has the YAML definition to associate the health check object with VMI by means of pod definitions.

Sample `hc.yaml` file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: healthcheck
---
apiVersion: core.contrail.juniper.net/v1alpha1
kind: ServiceHealthCheck
metadata:
  name: ping-hc
  namespace: healthcheck
spec:
  serviceHealthCheckProperties:
    delay: 2
    enabled: true
    healthCheckType: end-to-end
    maxRetries: 5
    monitorType: PING
    timeout: 5
---
apiVersion: core.contrail.juniper.net/v1alpha1
kind: ServiceHealthCheck
metadata:
  name: bfd-hc
  namespace: healthcheck
spec:
  serviceHealthCheckProperties:
    delay: 2
    enabled: true
    healthCheckType: link-local
    maxRetries: 5
    monitorType: BFD
    timeout: 5
---
apiVersion: core.contrail.juniper.net/v1alpha1
kind: ServiceHealthCheck
metadata:
  name: http-hc
  namespace: healthcheck
spec:
  serviceHealthCheckProperties:
```

```
    delay: 2
    enabled: true
    healthCheckType: end-to-end
    maxRetries: 5
    monitorType: HTTP
    timeout: 5
    httpMethod: GET
    expectedCodes: 200
    urlPath: /health
```

2. Complete the parameters to define the health check. Table 9 on page 49 lists and explains the parameters.

Table 9: Health Check Configurable Parameters

Field	Description
`Delay`	The delay, in seconds, to repeat the health check.
`DelayUsecs`	Time in micro seconds at which health check is repeated.
`Enabled`	Indicates that health check is enabled. The default is `False`.
`ExpectedCodes`	When the monitor protocol is HTTP, the expected return code for HTTP operations. Must be in the range of 200-299.
`HealthCheckType`	Indicates the health check type: `link-local`, `end-to-end`, `segment`, `vn-ip-list`, and `end2end`. The default is `link-local`. In both `link-local` and `end-to-end` modes, health check is executed for the pod on the vRouter where the VMI is running.
`HttpMethod`	When the monitor protocol is HTTP, the type of HTTP method used is GET.
`MaxRetries`	The number of retries to attempt before declaring an instance health down.

Table 9: Health Check Configurable Parameters *(Continued)*

Field	Description
`MonitorType`	The protocol type to be used: PING, BFD, or TCP.
`Timeout`	The number of seconds to wait for a response.
`TimeoutUsecs`	Time in micro seconds to wait for response.
`UrlPath`	Must be a valid URL. For example, `http://172.16.0.1/`*<path>*, The IP address can be a placeholder which will be replaced with the pod link-local IP address or metadata IP address.

Following is an abstract Golang schema for the health check resource.

```
type ServiceHealthCheckProperties struct {
    Delay              *int
    DelayUsecs         *int
    Enabled            boolean
    ExpectedCodes      int // Only for http
    HealthCheckType    (link-local | end-to-end |segment | vn-ip-list) //end2end
    HttpMethod          *string
    MaxRetries          int
    MonitorType         (ping | BFD |TCP)
    Timeout             int
    TimeoutUsecs        int

}

type ServiceHealthCheckSpec struct {
    ServiceHealthCheckProperties *ServiceHealthCheckProperties
}

type ServiceHealthCheckStatus struct {
    uuid *string
}

type ServiceHealthCheck struct {
    <kube_specific_objetcs>
    Spec     ServiceHealthCheckSpec
```

```
    Status   ServiceHealthCheckStatus
}
```

The YML representation for the Golang schema is:

````
```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: ServiceHealthCheck
metadata:
  name: ping-hc
  namespace: healthcheck
spec:
  serviceHealthCheckProperties:
    delay: 2
    enabled: true
    healthCheckType: end-to-end  #valid values are link-local|end-to-end|segment|vn-ip-list
    maxRetries: 5
    monitorType: PING  #valid are PING|HTTP|BFD
    timeout: 5
```
````

3. Link the health check object to the VMI by means of the pod annotation reference value `core.juniper.net/health-check`. The default behavior is to associate the health check with the primary interface.

```
apiVersion: v1
kind: Pod
metadata:
  name: hc_pod
  namespace: hc_ns
  annotations:
     core.juniper.net/health-check: '[{"name": "ping-hc", "namespace": "healthcheck"}]'
spec:
  <>
```

4. (Optional) To link the health check with multiple interfaces (attached to different NAD or VN), you can refer the health check object within the `cni-args` section. Following is an example of configured `cni-args` in annotations.

```
apiVersion: v1
kind: Pod
metadata:
  name: hc_pod
  namespace: hc_ns
  annotations:
          k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name": "hc-vn",
          "namespace": "healthcheck",
          "cni-args": {
            "core.juniper.net/health-check": "[{\"name\": \"ping-hc\", \"namespace\":
\"healthcheck\"}]"
          }
        }
      ]
spec:
  <>
```

Existing VMI objects will have a new field to reference the `HealthCheck` object.

```
type VirtualMachineInterfaceStatus struct {
    <existing_vmi_status_attributes>
    ServiceHealthCheckReference *ResourceReference
}


type VirtualMachineInterface struct {
    <other VMI attributes>
    Status   VirtualMachineInterfaceStatus
}
```

For the PING or HTTP monitoring-based health check minimum interval is **`1second`**. If you need a sub-second level health check for critical applications, you can opt for the BFD-based monitoring type.

Health Check Process

The Contrail vRouter agent is responsible for providing the health check service. The agent spawns a health check probe process to monitor the status of a service hosted on the same compute node. Then the process updates the status to the vRouter agent.

The vRouter agent acts on the status provided by the script to withdraw or restore the exported interface routes. The agent is responsible for providing a link-local metadata IP address for allowing the script to communicate with the destination IP address from the underlay network, using appropriate NAT translations. In a running system, this information is displayed in the vRouter agent introspect at:

```
http://<compute-node-ip>:8085/Snh_HealthCheckSandeshReq?uuid=
```

3

CHAPTER

Configure DPDK

Control Pod Scheduling on DPDK Nodes

SUMMARY

Cloud-Native Contrail Networking release 22.2 supports a custom plugin that schedules pods based on node interface capacity. This plugin is comprised of several APIs that filter and select optimal DPDK nodes for pod assignment.

IN THIS SECTION

Pod Scheduling in Kubernetes

In Kubernetes, a scheduler monitors newly-created pods for pods with no node assignment. The scheduler attempts to assign these pods to suitable nodes using a filtering phase and a scoring phase. Potential nodes are filtered based on attributes like the resource requirements of a pod. If a node doesn't have the available resources for a pod, that node is filtered out. If more than one node passes the filtering phase, Kubernetes scores and ranks the remaining nodes based on their suitability for a given pod. The scheduler assigns a pod to the node with the highest ranking. If two nodes have the same score, the scheduler picks a node at random.

Kubernetes Scheduling Framework Overview

The Kubernetes Scheduling Framework adds new scheduling APIs to the default cluster scheduler for extended scheduling functionality. The framework performs a scheduling cycle and a binding cycle for each pod. The scheduling cycle selects an optimal node for a pod, and the binding cycle applies that decision to the cluster. The scheduling and binding cycles expose several extension points during the course of their individual cycles. Plugins are registered to be called at various extension points. For example, during the scheduling cycle, one of the exposed extension points is called Filter. When the scheduling cycling reaches the Filter extension point, Filter plugins are called to perform filtering tasks.

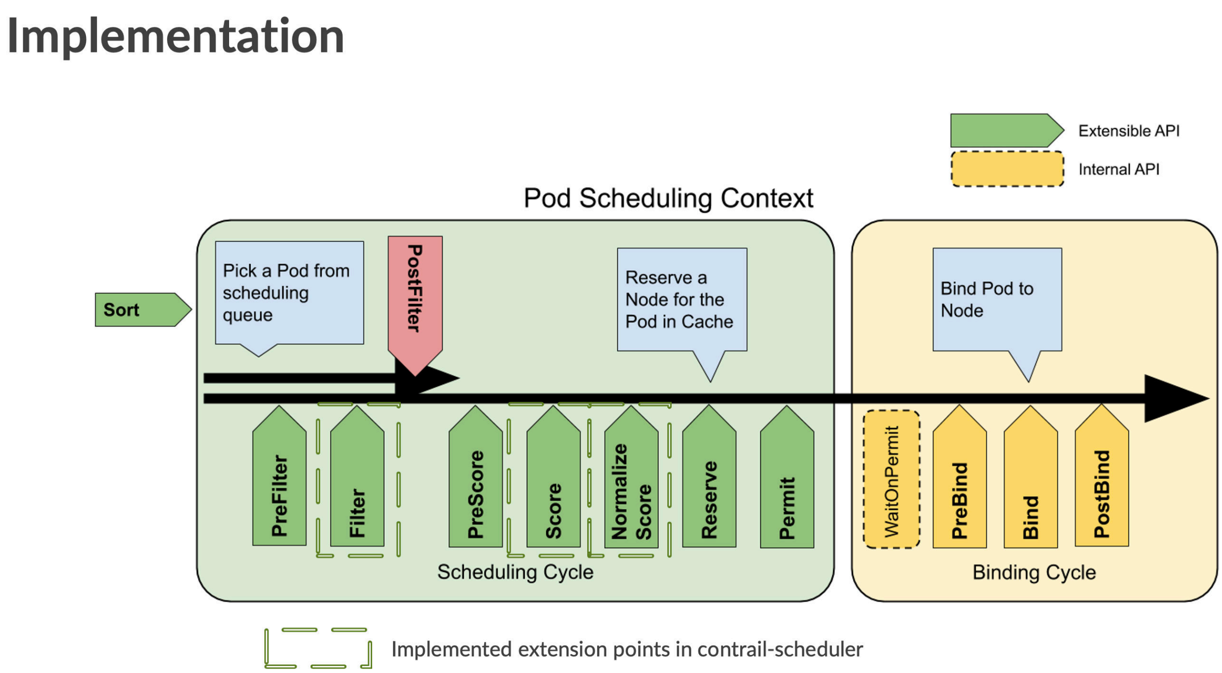
Contrail Custom Scheduler Overview

Cloud-Native Contrail Networking supports the deployment of DPDK nodes for high-throughput applications. DPDK nodes have a 32 VMI (Virtual Machine Interface) limit by default. This means that a DPDK node hosts a maximum of 32 pods. The Kubernetes default scheduler doesn't currently support a mechanism for recognizing DPDK node requirements and limitations. As a result, Cloud-Native Contrail Networking provides a custom scheduler built on top of the Kubernetes Scheduling Framework that implements a `VMICapacity` plugin to support pod scheduling on DPDK nodes.

Contrail Custom Scheduler Implementation in Cloud-Native Contrail Networking

Cloud-Native Contrail Networking Custom Scheduler supports a `VMICapacity` plugin which implements Filter, Score, and `NormalizeScore` extension points in the scheduler framework. See the sections below for more information about these extension points.

Figure 2: Contrail Custom Scheduler Extension Points



Filter

These plugins filter out nodes that cannot run the pod. Nodes are filtered based on VMI capacity. If a node has the maximum amount of allocated pods, that node is filtered out and the scheduler marks the pod as unusable on that node. Non-DPDK nodes are also filtered out in this phase based on user-configured `nodeLabels` that identify DPDK nodes.

Score

These plugins rank nodes that passed the filtering phase. The scheduler calls a series of scoring plugins for each node. In Cloud-Native Contrail Networking, a node's score is based on the number of VMIs currently active in the node. If only one node passes the Filter stage, the Score and `NormalizeScore` extension points are skipped and the scheduler assigns the pod to that node.

NormalizeScore

These plugins modify node scores before the scheduler computes a final ranking of nodes. The number of active VMIs on a node determines that node's score. The higher the number of active VMIs, the lower the score, and vice versa. The score is normalised in the range of 0-100. After the `NormalizeScore` phase, the scheduler combines node scores for all plugins according to the configured plugin weights defined in the scheduler configuration.

Deploy the Kubernetes Scheduling Framework as a Secondary Scheduluer

Follow these high-level steps to deploy the Contrail Custom Scheduler as a secondary scheduler that runs alongside your default Kubernetes scheduler:

1. Create configuration files for your custom scheduler.
2. Create a vanilla deployment with proper volume mounts and flags for your scheduler configurations.
3. Verify the pod(s) that you want the custom scheduler to schedule.

See the sections below for more information.

Create Configuration Files for Your Custom Scheduler

The custom scheduler requires a `kubeconfig` and a scheduler configuration file. Consider the following sample scheduler configuration file:

```
apiVersion: kubescheduler.config.k8s.io/v1beta3
clientConnection:
  acceptContentTypes: ""
  burst: 100
  contentType: application/vnd.kubernetes.protobuf
  kubeconfig: /tmp/config/kubeconfig
  qps: 50
enableContentionProfiling: true
enableProfiling: true
kind: KubeSchedulerConfiguration
leaderElection:
```

```
  leaderElect: false
profiles:
- pluginConfig:
  - args:
      apiVersion: kubescheduler.config.k8s.io/v1beta3
      kind: VMICapacityArgs
      maxVMICount: 32
      nodeLabels:
        agent-mode: dpdk
    name: VMICapacity
  plugins:
    filter:
      enabled:
      - name: VMICapacity
        weight: 0
    score:
      enabled:
      - name: VMICapacity
        weight: 0
  schedulerName: contrail-scheduler
```

Note the following fields:

- `schedulerName`: The name of the custom scheduler. This name must be unique to a cluster. You must define this field if you want a pod to be scheduled using this scheduler.
- `kubeconfig`: The path to the `kubeconfig` file mounted on the pod's filesystem.
- `maxVMICount`: The maximum number of VMIs a DPDK node accommodates.
- `nodeLabels`: A set of labels identifying a group of DPDK nodes.
- `VMICapacity`: The name of the plugin that enables Kubernetes to determine VMI capacity for DPDK nodes.

Create a Vanilla Deployment with Proper Volume Mounts and Flags for Your Scheduler Configurations

Ensure that you don't have more than one instance of a scheduler deployment running on a single node as this results in a port conflict. Use node affinity rules or a DaemonSet in order to run multiple instances of a scheduler on separate node in case of high availability (HA) requirements. Modify the scheduler configuration as needed in order to enable leader election. For more information about leader election, see the "Enable leader election section" of the following Kubernetes article: Configure Multiple Schedulers.

The following YAML file shows an example of a scheduler deployment:

NOTE: You must create a namespace for the scheduler before launching a scheduler deployment YAML. The scheduler operates under the namespace that your create.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: contrail-scheduler
  namespace: scheduler
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: contrail -scheduler
subjects:
- kind: ServiceAccount
  name: contrail -scheduler
  namespace: scheduler
roleRef:
  kind: ClusterRole
  name: system:kube-scheduler
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: contrail-scheduler
  namespace: scheduler
  labels:
    app: scheduler
spec:
  replicas: 1
  selector:
    matchLabels:
      app: scheduler
  template:
    metadata:
```

```
    labels:
      app: scheduler
  spec:
    serviceAccountName: contrail-scheduler
    securityContext:
      fsGroup: 2000
      runAsGroup: 3000
      runAsNonRoot: true
      runAsUser: 1000
    containers:
    - name: contrail-scheduler
      image: <scheduler-image>
      command:
      - /contrail-scheduler
      - --kubeconfig=/tmp/config/kubeconfig
      - --authentication-kubeconfig=/tmp/config/kubeconfig
      - --authorization-kubeconfig=/tmp/config/kubeconfig
      - --config=/tmp/scheduler/scheduler-config
      - --secure-port=<metrics-port; defaults to 10259>
      livenessProbe:
        failureThreshold: 8
        httpGet:
          path: /healthz
          port: <secure-port>
          scheme: HTTPS
        initialDelaySeconds: 10
        periodSeconds: 10
        timeoutSeconds: 15
      resources:
        requests:
          cpu: 100m
      readinessProbe:
        failureThreshold: 8
        httpGet:
          path: /healthz
          port: <secure-port>
          scheme: HTTPS
        initialDelaySeconds: 10
        periodSeconds: 10
        timeoutSeconds: 15
      volumeMounts:
      - mountPath: /tmp/config
        name: kubeconfig
```

```
          readOnly: true
        - mountPath: /tmp/scheduler
          name: scheduler-config
          readOnly: true
      hostNetwork: false
      hostPID: false
      volumes:
      - name: kubeconfig
        <volume for kubeconfig file>
      - name: scheduler-config
        <volume for scheduler configuration file>
```

Verify the Pod(s) That You Want the Custom Scheduler to Schedule

The following pod manifest shows an example of a pod deployment using the secondary scheduler:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  schedulerName: contrail-scheduler
  containers:
  - name: test
    image: busybox:latest
    command: ["/bin/sh","-c", "while true; do echo hello; sleep 10;done"]
```

Note the `schedulerName`. This field tells Kubernetes which scheduler to use when deploying a pod. You must define this field in each pod's manifest that you want deployed this way. A pod's deployment state remains pending if the specified scheduler is not present in the cluster.

4

CHAPTER

Configure Services

Display Microservice Status in Cloud-Native Contrail Networking

IN THIS SECTION

Juniper Cloud-Native Contrail® Networking™ supports microservices in environments using Contrail Networking Release 22.1 or later in a Kubernetes-orchestrated environment.

To display service status for the Contrail cluster, you need:

- CLI tool, such as `kubectl` to provide the overall system status of all the services running.
- The `contrailstatus` plugin must be installed along with `kubectl`.
- Use of command `kubectl contrailstatus` to request the status of various services.

Overview: Microservice Status in Cloud-Native Contrail Networking

Microservices exist as small, independent applications deployed without updating the entire Contrail Networking deployment and provides better ways to manage to the life cycle of containers. The containers and their processes are grouped as services and microservices.

ContrailStatus is a `kubectl` plugin used to display the status information of Contrail Networking services in the three different planes (configuration, control, and data). In addition to the usual containers in a specific service, `init` (initialization) container status within the service and the relative software status, such as BGP and XMPP in `control_controller` are also visible.

The `contrailstatus` plug-in is categorized into two sections:

- Deployment status
- Resource status

Display Microservice Status

The following outputs are examples showing deployment status updates and resource status updates to the pods for all planes.

Display Deployment Status

Display deployment status in either short or default form.

All Planes Deployment Status

To display the deployment status for all of the planes and request the short form:

```
root@helper ~] # kubectl contrailstatus -short


PLANE               STATUS
config              nok
control             ok
data                ok
```

The option `-short` for short form only displays output for the pod name and status. The following example outputs are using the default form.

Configuration Plane Deployment Status

To display the deployment status to the configuration plane:

```
root@helper ~] # kubectl contrailstatus deployment -p config


PODNAME                      STATUS           NODE       IP                 MESSAGE
apiserver-86885bf7d8-q27qk    nok             node       10.1.1.1          process not up,
init cont…..
apiserver-86885bf7d8-sdsdd    ok              node2      10.1.1.2
apiserver-86885bf7d8-sdsss    ok              node3      10.1.1.3
controller-6998bd846f-5cgf7   ok              node1      10.1.1.1
controller-6998bd846f-5cgf8   ok              node2      10.1.1.2
controller-6998bd846f-5cg10   nok             node3      10.1.1.3          o/1 node is not
allocated.
```

```
cluster1-kubemanager-7cff895-sdfsd ok          node2       10.1.1.2
cluster1-kubemanager-7cff895-sdfsa ok          node3       10.1.1.3
```

Data Plane Deployment Status

To display the deployment status to the data plane:

```
root@helper ~] # kubectl contrailstatus deployment -p data


PODNAME                             STATUS           NODE        IP               MESSAGE
vrouter-86885bf7d8-q27qk            nok              node        10.1.1.1          process
not up, init cont.…..
vrouter-86885bf7d8-sdsdd            ok               node2       10.1.1.2
```

Control Plane Deployment Status

To display the deployment status to the control plane:

```
root@helper ~] # kubectl contrailstatus deployment -p control


PODNAME                          STATUS          NODE        IP               MESSAGE
contrail-control-0               nok             node        10.1.1.1          process not
up, init cont.…..
contrail-control-1               ok              node2       10.1.1.2
```

Display Resource Status

The `contrailstatus` plugin also displays status updates for deployment resources, such as XMPP and BGP.

Data Plane Resource Status

To display the resource status of `bgprouter` to the data plane:

```
root@helper ~] kubectl contrailstatus resource bgprouter


PODNAME     STATUS     SERVICE
```

```
bgprouter1  nok        xmpp, bgp not working/has error..
bgprouter2  nok
bgprouter2 ok
```

Control Node Resource Status

To display the resource status in the control node, run the following command. The command gives the output for the XMPP session.

```
root@helper ~] kubectl contrailstatus resource bgprouter -s xmpp


LOCAL           NEIGHBOR          STATE              POD
bgprouter1      vr1               established (ok)   contrail-control-0
bgprouter1      vr2               active (nok)       contrail-control-0
bgprouter2      vr1                                  contrail-control-1
bgprouter2      vr3                                  contrail-control-1
```

To display the resource status in the control node, run the following command. The command gives the output for the BGP session.

```
root@helper ~]  kubectl contrailstatus resource bgprouter -s bgp


LOCAL          NEIGHBOR         STATE             POD
bgprouter1     bgprouter2       established (ok)  contrail-control-0
bgprouter1     bgprouter3       active (nok)      contrail-control-0
bgprouter2     bgprouter1       established (ok)  contrail-control-1
bgprouter2     bgprouter3       established (ok)  contrail-control-1
```

All Planes Resource Status

To display the resource status on all of the planes:

```
[root@helper ~] # kubectl contrailstatus -all


NAME                         STATUS    PLANE     ERRORNOTES
apiserver-86789f7d8-q37qf    Active    Config
```

```
NAME                        STATUS    PLANE     ERRORNOTES
control-1                   Active    control
BGP-1                       Active    control
XMPP-1                      Active    control


NAME                        STATUS    PLANE     ERRORNOTES
vrouter-86789f7d8-q37qk     Active    data


[root@helper ~] #
```

Services Status for Multiple Nodes

The following (same) command displays the status of various services running on multiple nodes in a cluster. If the running controller is active without any errors, the status column next to the service displays as `Active`. If the controller has any error, the status column of the controller displays as `Not-Active`. The output includes the status of various controllers and containers in the controllers.

To display the status of various services running on multiple nodes in a cluster:

```
[root@helper ~] # kubectl contrailstatus -all

NAME                                 STATUS     ERRORNOTES
apiserver-86885bf7d8-q27qk           Active
apiserver-86885bf7d8-sdsdd           Active
apiserver-86885bf7d8-sdsss           Active
controller-6998bd846f-5cgf7          Active
controller-6998bd846f-5cgf8          Active
controller-6998bd846f-5cg10          Active
cluster1-kubemanager-7cff895-sdfsd   Active
cluster1-kubemanager-7cff895-sdfsa   Active


NAME            STATUS   ERRORNOTES
control-1       Active
control-2       Active
control-3       Active
BGP-1           Active
BGP-2           Active
XMPP-1          Active
Xmpp-2          Active
```

```
NAME                        STATUS   ERRORNOTES
vrouter-86789f7d8-q37qk     Active
vrouter-8905bf7d8-q47qk     Active
vrouter-8688bf7d8-q57qk     Active


[root@helper ~] #
```

NodePort Service Support in Cloud-Native Contrail Networking

IN THIS SECTION

Juniper Networks supports Kubernetes NodePort service in environments using Cloud-Native Contrail® Networking™ Release 22.1 or later in a Kubernetes-orchestrated environment.

In Kubernetes, a service is an abstraction that defines a logical set of pods and the policy, by which you (the administrator) can access the pods. The set of pods implementing a service is selected based on the `LabelSelector` object in the service definition. NodePort service exposes a service on each node's IP at a static port. It maps the static port on each node with a port of the application on the pod.

In Contrail Networking, Kubernetes NodePort service is implemented using the `InstanceIP` resource and `FloatingIP` resource, both of which are similar to the `ClusterIP` service.

Kubernetes provides a flat networking model in which all pods can talk to each other. Network policy is added to provide security between the pods. Contrail Networking integrated with Kubernetes adds additional networking functionality, including multi-tenancy, network isolation, micro-segmentation with network policies, and load balancing.

The following table lists the mapping between Kubernetes concepts and Contrail Networking resources.

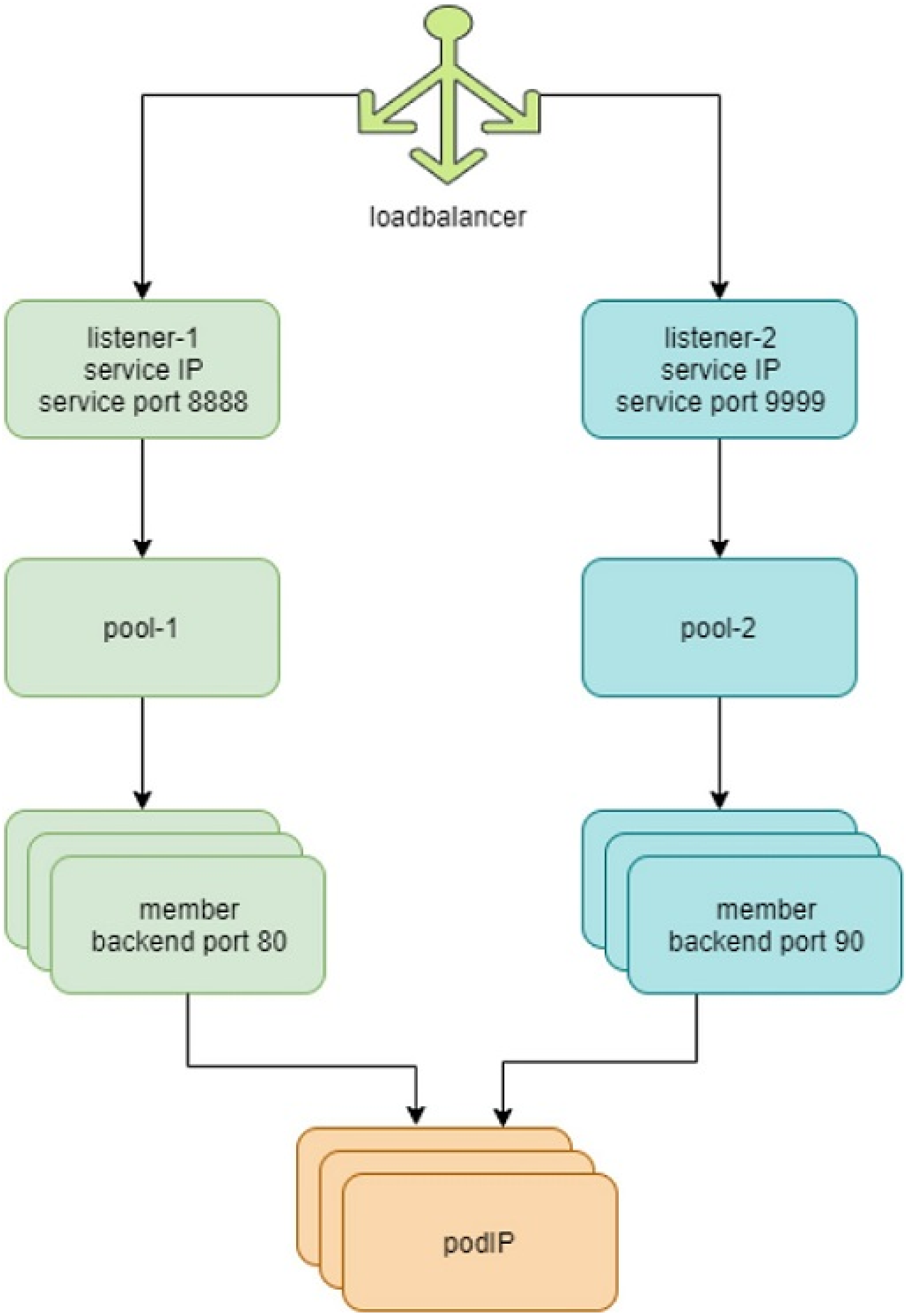
Table 10: Kubernetes Concepts to Contrail Networking Resource Mapping

Kubernetes Concept	Contrail Networking Resource
Namespace	Shared or single project
Pod	Virtual Machine
Service	Equal-cost multipath (ECMP) LoadBalancer
Ingress	HAProxy LoadBalancer for URL
Network Policy	Contrail Security

Contrail Networking Load Balancer Objects

Figure 3 on page 71 and the following list describe the load balancer objects in Contrail Networking.

Figure 3: Load Balancer Objects



- Each service in Contrail Networking is represented by a load balancer object.
- For each service port, a listener object is created for the same service load balancer.

- For each listener there is a pool object.
- The pool contains members. Depending on the number of backend pods, one pool might have multiple members.
- Each member object in the pool maps to one of the backend pods.
- The `contrail-kube-manager` listens to `kube-apiserver` for the Kubernetes service. When a service is created, a load balancer object with `loadbalancer_provider` type `native` is created.
- The load balancer has a virtual IP address (VIP), which is the same as the service IP address.
- The service IP/VIP address is linked to the interface of each backend pod. This is accomplished with an ECMP load balancer driver.
- The linkage from the service IP address to the interfaces of multiple backend pods creates an ECMP next hop in Contrail Networking. Traffic is load balanced from the source pod directly to one of the backend pods.
- The `contrail-kube-manager` continues to listen to `kube-apiserver` for any changes. Based on the pod list in endpoints, `contrail-kube-manager` knows the most current backend pods and updates members in the pool.

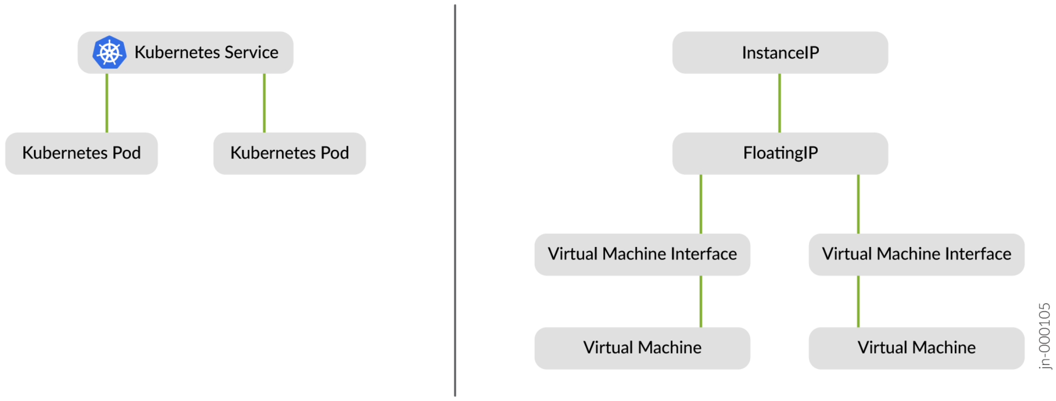
NodePort Service in Contrail Networking

A controller service is implemented in `kube-manager`. The `kube-manager` is the interface between Kubernetes core resources, such as service and the extended Contrail resources, such as `VirtualNetwork` and `RoutingInstance`. This controller service watchs events going through the resource endpoints. An endpoint receives an event for any change related to its service. The endpoint also receives an event for pods created and deleted that match the service selector. The controller service handles creating the Contrail resources needed: See Figure 4 on page 73.

- `InstanceIP` resource related to the `ServiceNetwork`
- `FloatingIP` resource and the associated `VirtualMachineInterfaces`

When you create a service, an associated endpoint is automatically created by Kubernetes, which allows the controller service to receive new requests.

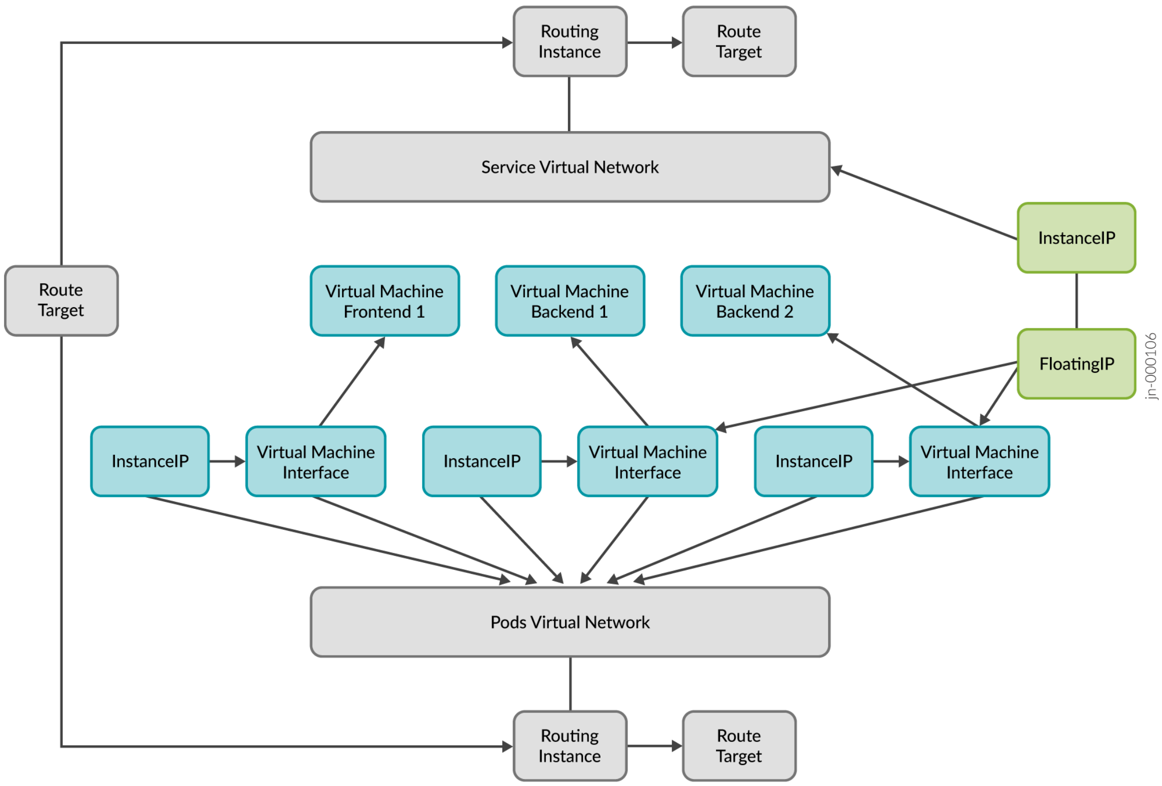
Figure 4: Controller Service Creates Contrail Resources



Workflow of Creating NodePort Service

Figure 5 on page 74 and the steps following detail the workflow when NodePort service is created.

Figure 5: Creating NodePort Service



1. When the NodePort service is created, `InstanceIP` (IIP) is created. The `InstanceIP` resource specifies a fixed IP address and its characteristics that belong to a subnet of a referred virtual network.

2. Once the endpoint is connected to the NodePort service, the `FloatingIP` is created.The `kube-manager` watches for the creation of endpoints connected to a service.

3. When a new endpoint is created, `kube-manager` then creates an `InstanceIP` in the `ServiceVirtualNetwork` subnet. The `kube-manager` then creates a `FloatingIP` using the `InstanceIP` as the parent.

4. The `FloatingIP` resource specifies a special kind of IP address that does not belong to a specific `VirtualMachineInterface` (VMI). The `FloatingIP` is assigned from a separate `VirtualNetwork` subnet and can be associated with multiple VMIs. When associated with multiple VMIs, traffic destined to the `FloatingIP` is distributed using ECMP across all VMIs.

Notes about VMIs:

- VMIs are dynamically updated as pods and labels are added and deleted.

- A VMI represents an interface (port) into a virtual network and might or might not have a corresponding virtual machine.

- A VMI has at minimum a MAC address and an IP address.

Notes about VMs:

- A VM resource represents a compute container. For example VM, baremetal, pod, or container.
- Each VM can communicate with other VMs on the same tenant network, subject to policy restrictions.
- As tenant networks are isolated, VMs in one tenant cannot communicate with VMs in another tenant unless specifically allowed by policy.

Kubernetes Probes and Kubernetes NodePort Service

The `kubelet`, an agent that runs on each node, needs reachability to pods for liveness and readiness probes. Contrail network policy is created between the IP fabric network and pod network to provide reachability between node and pods. Whenever the pod network is created, the network policy is attached to the pod network to provide reachability between node and pods. As a result, any process in the node can reach the pods.

Kubernetes NodePort service is based on node reachability to pods. Since Contrail Networking provides connectivity between nodes and pods through the Contrail network policy, NodePort is supported.

NodePort service supports two types of traffic:

- East-West
- Fabric to Pod

NodePort Service Port Mapping

The port mappings for Kubernetes NodePort service are located in the `FloatingIp` resource in the YAML file. In `FloatingIp`, the ports are added in `"floatingIpPortMappings"`.

If the `targetPort` is not mentioned in the service, then the `port` value is specified as default.

Following is an example `spec` YAML file for NodePort service with port details:

```
spec:
 clusterIP: 10.100.13.106
 clusterIPs:
```

```
  - 10.100.13.106
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: my-nginx
  sessionAffinity: None
```

For the above example `spec` YAML, `"floatingIpPortMappings"` are created in the `FloatingIp` resource:

Example `"floatingIpPortMappings"` YAML:

```
"floatingIpPortMappings": {
            "portMappings": [
                {
                    "srcPort": 80,
                    "dstPort": 80,
                    "protocol": "TCP"
                }
            ]
        }
```

Example: NodePort Service Request Journey

Let's follow the journey of a NodePort service request from when the request gets to the node port until the service request reaches the backend pod.

Nodeport service relies on `kubeproxy`. The Kubernetes network proxy (`kube-proxy`) is a daemon running on each node. It reflects the services defined in the cluster and manages the rules to load balance requests to a service's backend pods.

In the following example, the NodePort service `apple-service` is created and its endpoints are associated.

```
user@domain ~ % kubectl describe svc apple-service
Name:                     apple-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=apple
```

```
Type:                      NodePort
IP Families:               <none>
IP:                        10.105.135.144
IPs:                       10.105.135.144
Port:                      <unset>  5678/TCP
TargetPort:                5678/TCP
NodePort:                  <unset>  31050/TCP
Endpoints:                 10.244.0.4:5678
Session Affinity:          None
External Traffic Policy:   Cluster
Events:                    <none>

user@domain ~ % kubectl get endpoints apple-service
NAME            ENDPOINTS         AGE
apple-service   10.244.0.4:5678   2d18h
```

Each time a service is created, deleted, or the endpoints are modified, `kube-proxy` updates the `iptables` rules on each node of the cluster. View the `iptables` chains to understand and follow the journey of the request.

First, the KUBE-NODEPORTS chain takes into account the packets coming on service of type `NodePort`.

```
$ sudo iptables -L KUBE-NODEPORTS -t nat
Chain KUBE-NODEPORTS (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  tcp  --  anywhere             anywhere             /* default/apple-service */
tcp dpt:31050
KUBE-SVC-Y4TE457BRBWMNDKG  tcp  --  anywhere             anywhere             /* default/apple-
service */ tcp dpt:31050
```

Each packet coming into port `31050` is first handled by the KUBE-MARK-MASQ, which tags the packet with a `0x4000` value.

Next, the packet is handled by the KUBE-SVC-Y4TE457BRBWMNDKG chain (referenced in the KUBE-NODEPORTS chain above). If we take a closer look at that one, we can see additional `iptables` chains:

```
$ sudo iptables -L KUBE-SVC-Y4TE457BRBWMNDKG -t nat
Chain KUBE-SVC-Y4TE457BRBWMNDKG (2 references)
target     prot opt source               destination
KUBE-SEP-LCGKUEHRD52LOEFX  all  --  anywhere             anywhere             /* default/apple-
service */
```

Inspect the KUBE-SEP-LCGKUEHRD52LOEFX chains to see that they define the routing to one of the backend pods running the `apple-service` application.

```
$ sudo iptables -L KUBE-SEP-LCGKUEHRD52LOEFX -t nat
Chain KUBE-SEP-LCGKUEHRD52LOEFX (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  10.244.0.4           anywhere             /* default/apple-service */
DNAT       tcp  --  anywhere             anywhere             /* default/apple-service */ tcp
to:10.244.0.4:5678
```

This completes the journey of a NodePort service request from when the request gets to the node port until the service request reaches the backend pod.

Local Option Limitation in External Traffic Policy

NodePort service with `externalTrafficPolicy` set as `Local` is not supported in Contrail Networking Release 22.1.

The `externalTrafficPolicy` denotes if this service desires to route external traffic to node-local or cluster-wide endpoints.

- `Local` preserves the client source IP address and avoids a second hop for NodePort type services.
- `Cluster` obscures the client source IP address and might cause a second hop to another node.

`Cluster` is the default for `externalTrafficPolicy`.

Update or Delete a Service, or Remove a Pod from Service

- Update of service—Any modifiable fields can be changed, excluding `Name` and `Namespace`. For example, Nodeport service can be changed to `ClusterIp` by changing the `Type` field in the service YAML definition.
- Deletion of service—A service, irrespective of `Type`, can be deleted with the command:

  `kubectl delete -n <name_space> <service_name>`
- Removing pod from service—This can be achieved by changing the `Labels` and `Selector` on the service or pod.

Create a LoadBalancer Service

SUMMARY

This topic describes how to create a Load Balancer service in Juniper Cloud-Native Contrail® Networking™ (CN2).

IN THIS SECTION

LoadBalancer Service Overview

Juniper Networks supports LoadBalancer services using Cloud-Native Contrail Networking (CN2) Release 22.1 or later in a Kubernetes-orchestrated environment.

In Kubernetes, a service is an abstract way to expose an application running on a set of pods as a network service. See Kubernetes Services.

In CN2, the Kubernetes LoadBalancer service is implemented using the `InstanceIP` resource and `FloatingIP` resource, both of which are similar to the `ClusterIP` service.

- The `FloatingIP` is used in the service implementation to expose an external IP to the LoadBalancer service. The `FloatingIP` resource is also associated with the pod's `VirtualMachineInterfaces`.
- The `InstanceIP` resource is related to the `VirtualNetwork`. Two `instanceIPs` are created, one for the service network and one for the external network.

A controller service is implemented in Contrail's kubemanger. Kubemanager is the interface between Kubernetes core resources, and the extended Contrail resources, such as the `VirtualNetwork`. When you create a LoadBalancer service, kubemanager listens and allocates the IP from an external virtual network. This external virtual network exposes the LoadBalancer service on the external IPs. Any requests received through the provisioned external IP is ECMP load-balanced across the pods associated with the LoadBalancer.

Create a LoadBalancer Service

IN THIS SECTION

The following sections describe how to create a LoadBalancer service in CN2.

Prerequisites

Before you create a LoadBalancer service, make sure of the following:

- You have set up a working cloud networking environment using Kubernetes orchestration.
- Cloud-Native Contrail Networking is operational.
- You configured kubemanager to define the external networks to be used by the LoadBalancer service.

Define an External Virtual Network

Before you create a LoadBalancer service, you must define an external virtual network. You can define the virtual network two ways, by creating a *NetworkAttachmentDefinition* or by creating a virtual network.

NOTE: A Multus deployment requires that you only use a *NetworkAttachmentDefinition* to define an external network.

The following example illustrates how to define an external virtual network using a `NetworkAttachmentDefinition`. In this example, the external IP is allocated from the subnet range `192.168.102.0/24`.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
 name: ecmp-default
 namespace: ecmp-project
```

```
 annotations:
   juniper.net/networks: '{
   "ipamV4Subnet": "192.168.102.0/24",
   "fabricSNAT": false
   "core.juniper.net/display-name: "External Virtual Network"
 }'
 core.juniper.net/display-name: "External Virtual Network"
labels:
  service.contrail.juniper.net/externalNetworkSelector: default-external
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "ecmp-default",
    "type": "contrail-k8s-cni"
}'
```

When you apply the `NetworkAttachmentDefinition`, kubemanger creates a virtual network with the name `ecmp-default` in the namespace `ecmp-project`.

Specify the External Networks

By default, kubemanager allocates the external IP for a LoadBalancer service from the `default-external` network. To allocate the external IP from a different network, you must define the external network using selectors.

The following is an example of a Kubemanager.yaml file specifying the `default-external` network selector and user-defined network selectors.

```
apiVersion: configplane.juniper.net/v1alpha1
kind: Kubemanager
metadata:
 generation: 148
 name: contrail-k8s-kubemanager
 namespace: contrail
spec:
 externalNetworkSelectors:
  default-external:
   networkSelector:
    matchLabels:
     service.contrail.juniper.net/externalNetwork: default-external
 custom-external:
  namespaceSelector:
```

```
    matchLabels:
     customNamespaceKey: custom-namespace-value
  networkSelector:
   matchLabels:
    customNetworkKey: custom-network-value
  custom-external-in-service-namespace:
   networkSelector:
    matchLabels:
     customExternalInServiceNetworkKey: custom-external-in-service-network-value
```

The VirtualNetworks listed below match the labels that shown in the Kubemanager.yaml above (in relative order).

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualNetwork # matches example 1
metadata:
  name: default-external-vn
  namespace: contrail
  labels:
    service.contrail.juniper.net/externalNetworkSelector: default-external
spec:
  v4SubnetReference:
    apiVersion: core.contrail.juniper.net/v1alpha1
    attributes:
      ipamSubnets:
      - defaultGateway: 10.244.0.1
        enableDHCP: true
        subnet:
          ipPrefix: 10.244.0.0
          ipPrefixLen: "16"

---#this is how you define namespace selector
# Namespace must have appropriate label if required by namespaceSelector
apiVersion: v1
kind: Namespace
metadata:
  labels:
      customNamespaceKey: custom-namespace-value #user for your external ip
  name: contrail
  namespace: custom-namespace
---
apiVersion: core.contrail.juniper.net/v1alpha1
```

```
kind: VirtualNetwork
metadata:
  name: external-vn-1 # matches example 2 and example 3
  namespace: custom-namespace
  labels:
    customNetworkKey: custom-network-value
spec:
  v4SubnetReference:
    apiVersion: core.contrail.juniper.net/v1alpha1
    attributes:
      ipamSubnets:
      - defaultGateway: 10.0.0.1
        enableDHCP: true
        subnet:
          ipPrefix: 10.0.0.0
          ipPrefixLen: "16"

---

apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualNetwork
metadata:
  name: external-vn-2 # matches example 4
  namespace: custom-namespace
  labels:
    customExternalInServiceNetworkKey: custom-external-in-service-network-value
spec:
  v4SubnetReference:
    apiVersion: core.contrail.juniper.net/v1alpha1
    attributes:
      ipamSubnets:
      - defaultGateway: 192.168.0.1
        enableDHCP: true
        subnet:
          ipPrefix: 192.168.0.0
          ipPrefixLen: "16"
```

Define Service Level Annotations

Additionally, you can define the following service level annotations for external network discovery.

Annotation: externalNetwork .

In this example, the `externalNetwork` annotation allocates an external IP from the `evn` virtual network in the namespace `ns`.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  annotation:
    service.contrail.juniper.net/externalNetwork: ns/evn
spec:
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Annotation: externalNetworkSelector

In this example, the `externalNetworkSelector` matches the name of the `externalNetworkSelector` defined in kubemanager.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  annotation:
    service.contrail.juniper.net/externalNetworkSelector: custom-external
spec:
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

NOTE: You can also define service level annotations in the namespace of the Kubernetes cluster, or in the namespace of the Contrail cluster. The service-level annotations takes precedence.

Examples: External Network Selection

NOTE: The virtual networks defined in "Specify the External Networks" on page 81 are linked to the annotations in the following examples.

The external virtual network is selected from one of the following in priority order:

Example 1: Default Selector.

Kubemanager first looks for the default external network. This example uses the `default-external` selector because no annotation is specified.

Matches the network `contrail/default-external-vn`.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Example 2: Custom namespace

Matches the network `custom-namespace/external-vn-1`.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
```

```
  annotation:
    service.contrail.juniper.net/externalNetwork: custom-namespace/external-vn-1
spec:
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Example 3: External network matching preconfigured selector in a namespace.

Matches the network `custom-namespace/external-vn-1`.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  annotation:
    service.contrail.juniper.net/externalNetworkSelector: custom-external
spec:
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Example 4: External network matching preconfigured selector in service namespace.

Matches the network `custom-namespace/external-vn-2`.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: custom-namespace
  annotation:
    customExternalInServiceNetworkKey: custom-external-in-service-network-value
spec:
```

```
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Dual-Stack Networking Support

As an Administrator, you might need to select the IP family (IPv4 or IPv6) to use when defining a service. IPv4/IPv6 dual-stack networking enables the allocation of both IPv4 and IPv6 addresses to pods and services. If you do not define the IP family, the default IPv4 is used.

In this example, an IPv4 and IPv6 default external network is allocated for the LoadBalancer service.

```
apiVersion: v1
kind: Service
metadata:
  name: MyService
specs:
  ipFamilies: ["IPv4", "IPv6"]
```

For more information, see Overview: IPv4 and IPv6 Dual-Stack Networking.

Configure LoadBalancer Services without Selectors

In Kubernetes, you can expose an application running on a set of pods as a network service. Kubernetes uses selectors to automatically create a LoadBalancer service, but only uses the default primary interface for load balancing.

Starting in Cloud-Native Contrail Networking (CN2) Release 22.3, you can load-balance a service across multiple secondary interfaces. You create secondary interfaces in CN2 without using a selector. Because the LoadBalancer service has no selector, you must create the endpoint manually.

To create a secondary interface for a LoadBalancer service:

1. Create two virtual networks using a *NetworkAttachmentDefinition*.

The following example shows two networks. One network for the pod's secondary interface (**pod-subnet**) and another network (**lb-subnet**) for the LoadBalancer service external IP. These networks are connected by a common route target which routes traffic between the two networks.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
 name: pod-subnet
 namespace: my-lb
 annotations:
   juniper.net/networks: '{
     "ipamV4Subnet": "172.16.12.0/24",
     "routeTargetList": ["target:64521:1164"]
 }'
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "pod-subnet",
    "type": "contrail-k8s-cni"
}'
---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: lb-subnet
  namespace: my-lb
  annotations:
    juniper.net/networks: '{
      "ipamV4Subnet": "172.16.13.0/24",
      "routeTargetList": ["target:64521:1164"]
    }'
spec:
  config: '{
  "cniVersion": "0.3.1",
  "name": "lb-subnet",
  "type": "contrail-k8s-cni"
}'
```

2. Create the pods you want to load-balance the service on. You can create multiple pods.

In this example, we'll create two pods in the **my-lb** namespace, each with their own IP address.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: my-lb
  labels:
    run: ecmp
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
        {
          "name": "pod-subnet",
          "namespace": "my-lb",
          "ips": ["172.16.23.0"]
        }
      ]'
spec:
  containers:
    - name: front01-multiintf
      image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/google-containers/
toolbox
      command:
        ["bash", "-c", "ip route add 172.16.23.0/24 via 172.16.23.1 dev eth1;while true; do 
echo front01 | nc -w 1 -l -p 8080; done"]
      securityContext:
        privileged: true
---
apiVersion: v1
kind: Pod
metadata:
  name: my-pod1
  namespace: my-lb
  labels:
    run: ecmp
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
        {
         "name": "pod-subnet",
          "namespace": "my-lb",
          "ips": ["172.16.24.0"]
              }
```

```
      ]'
spec:
  containers:
    - name: front02-multiintf
      image: svl-artifactory.juniper.net/atom-docker/cn2/bazel-build/dev/google-containers/
toolbox
      command:
        ["bash", "-c", "ip route add 172.16.24.0/24 via 172.16.23.1 dev eth1; while true; do
echo front02 | nc -w 1 -l -p 8080; done"]
      securityContext:
        privileged: true
; done"]
```

3. Create a LoadBalancer service.

   In this example, we'll create a LoadBalancer service (**service-lb**) in the **my-lb** namespace. Note that this Service.YAML is not using a selector.

```
kind: Service
metadata:
  name: service-lb
  namespace: my-lb
  annotations:
    service.contrail.juniper.net/externalNetwork: my-lb/lb-subnet
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

4. Specify the endpoints (IP addresses) that you want to load balance the service on. In this example, two pod endpoints are specified for the secondary interfaces (ip: 172.16.23.0 and ip : 172.16.24.0).

   Make sure that the endpoint has the same name as the LoadBalancer service.

```
apiVersion: v1
kind: Endpoints
metadata:
  name: service-lb
  namespace: my-lb
```

```
subsets:
  - addresses:
      - ip: 172.16.23.0
      - ip: 172.16.24.0
    ports:
      - port: 8080
```

Success! You can now load balance a service across multiple pods with the secondary interface.

In addition to creating a LoadBalancer service on the secondary interface, you can use a selector to create a LoadBalancer service on the default primary interface. The primary interface can work in tandem with the secondary interface. You can use either interface to load balance across your desired service.

5

CHAPTER

Analytics

Contrail Networking Analytics

IN THIS SECTION

Overview: Analytics

Analytics is an optional feature set in Juniper Cloud-Native Contrail® Networking™ Release 22.1. It is packaged separately from the Contrail Networking core CNI components. Analytics also has its own installation procedure. The package consists of a combination of open-source software and Juniper developed software.

The analytics features are categorized into the following high-level functional areas:

- Metrics—Statistical time series data collected from the Contrail Networking components and the base Kubernetes system.
- Flow and Session Records—Network traffic information collected from the Contrail Networking vRouter.
- Sandesh User Visible Entities (UVE)—Records representing the system-wide state of externally visible objects that are collected from the Contrail Networking vRouter and control node components.
- Logs—Log messages collected from Kubernetes pods.
- Introspect—A diagnostic utility that provides an ability to browse the internal state of the Contrail Networking components.

Metrics

Data Model

Metric information is based on a numerical time series data model. Each data point in a series is a sample of some system state that gets collected at a regular interval. A sampled value is recorded along with a timestamp at which the collection occurred. A sample record can also contain an optional set of key-value pairs called labels. Labels provide a dimension capability for metrics where a given combination of labels for the same metric name identifies a particular dimensional instantiation of that metric. For example, a metric named `api_http_requests_total` can utilize labels in order to provide visibility into the request counts at a URL and method type level. In the following example, the metric record for a sample value of 10 will include a set of labels that indicate the type of request.

```
api_http_requests_total{method="POST", handler="/messages"} 10
```

Metric Data Types

Although all metric sample values are just numbers, there is a concept of type within this numerical data model. A metric is considered to be one of the following types:

- Counter—A cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart.
- Gauge—A metric that represents a single numerical value that can arbitrarily go up and down.
- Histogram—A histogram samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. The histogram also provides a sum of all observed values.
- Summary—Similar to a histogram, a summary samples observations (usually things like request durations and response sizes). While it also provides a total count of observations and a sum of all observed values, the summary calculates configurable quantiles over a sliding time window.

The metric functionality in Contrail Networking is implemented by Prometheus. For additional details about the metric data model, see the documentation at Prometheus.

Supported Metrics

The set of metrics supported by the analytics solution are categorized as shown below:

- "Contrail Networking Metric List" on page 99—Metrics collected from the vRouter and control node components.

- "Kubernetes Metric List" on page 113—Metrics collected from various Kubernetes components, such as `apiserver`, `etcd`, `kubelet`, and so on.
- "Cluster Node Metric List" on page 152—Host-level metrics collected from the Kubernetes cluster nodes.

Alerts

Alerts are generated based on an analysis of collected metric data. Every supported alert type is based on a rule definition that contains the following information:

- Alert Name—A unique string identifier for the alert type.
- Condition Expression—A Prometheus query language expression that gets evaluated against collected metric values in order to determine if the alert condition exists.
- Condition Duration—The amount of time the problematic condition has to exist in order for the alert to be generated.
- Severity—The alert level (critical, major, warning, info).
- Summary—A short description of the problematic condition.
- Description—A detailed description of the problematic condition.

The Contrail Networking analytics solution installs a set of "predefined alert rules" on page 169. You can also define your own custom alert rules. This is supported by the creation of PrometheusRule Kubernetes resources in the namespace where the analytics Helm chart is deployed. Following is an example of a custom alert rule.

```
apiVersion: monitoring.coreos.com/v1
  kind: PrometheusRule
  metadata:
    name: acme-corp-rules
  spec:
    groups:
    - name: acme-corp.rules
      rules:
      - alert: HostUnusualNetworkThroughputOut
        expr: "sum by (instance) (rate(node_network_transmit_bytes_total[2m])) / 1024 / 1024 >
100"
        labels:
          severity: warning
```

```
        annotations:
          summary: "Host unusual network throughput out (instance {{ $labels.instance }})"
          description: "Host network interfaces are sending too much data (> 100 MB/s)\n  VALUE
= {{ $value }}"
```

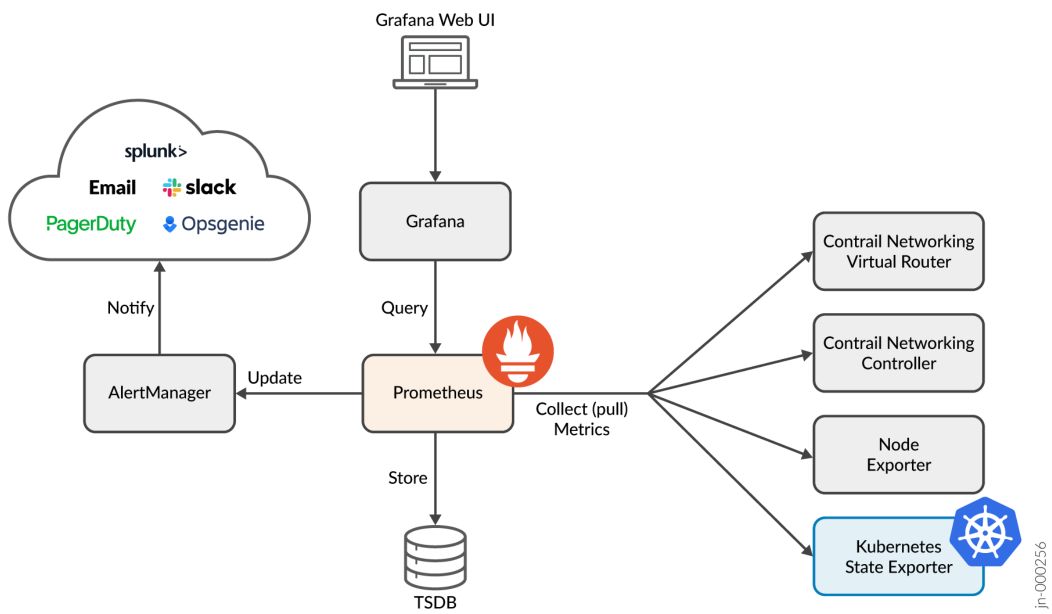
Generated alerts are stored as records in Prometheus and can be viewed in the Grafana UI. The AlertManager component supports integration with external systems, such as PagerDuty, OpsGenie, or email for alert notification.

Architecture

As shown in Figure 6 on page 97, Prometheus is the core component of the metrics architecture. Prometheus implements the following functionality:

- Collection—A periodic polling mechanism that invokes API calls against other components (exporters) to pull values for a set of metrics.
- Storage—A time series database that provides persistence for the metrics collected from the exporters.
- Query—An API supporting an expression language called PromQL (Prometheus query language) that allows the historical metric information to be retrieved from the database.
- Alerting—A framework providing an ability to define rules that produce alerts when certain conditions are observed in the collected metric data.

Figure 6: Metrics Architecture



The other components of the metrics architecture are:

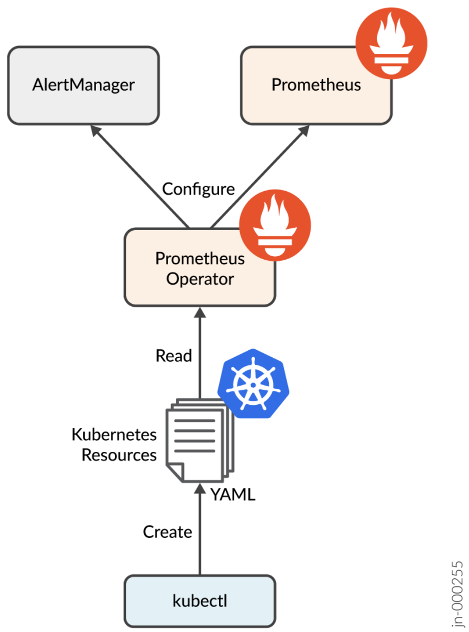
- Grafana—A service that provides a Web UI interface allowing the user to visualize the metric data in graphs.
- AlertManager—An integration service that notifies external systems of alerts generated by Prometheus.

Configuration

The metrics functionality does not require any configuration by the end-user. The installation of analytics takes care of configuring Prometheus to collect from the set of exporters that provide all of the metrics described in "Supported Metrics" on page 94. A group of default alerting rules is also automatically setup as part of the installation. This base functionality, however, can be extended by the administrator through additional configuration after the installation. For example, customer-specific alerting rules can be defined and the AlertManager can be configured to integrate with any of the supported external systems present in your environment.

The configuration of Prometheus and AlertManager involves an additional architectural component called the Prometheus Operator. As shown in Figure 7 on page 98, configuration is specified as Kubernetes custom resources. The operator is responsible for translating the contents of these resources into the native configuration understood by the Prometheus components and updating the components accordingly. Then taking care of restarting the components whenever a particular configuration change requires a restart.

Figure 7: Prometheus Operator



Documentation for the full set of resources supported by the operator is available at Prometheus Operator API. It is recommended that customers limit their configurations to the subset of resource types related to alert rule definition and external system integration.

Grafana

The main UI for viewing metric data and alerts is Grafana. The Grafana service is setup and automatically configured with Prometheus as a data source as part of the analytics installation. A set of default dashboards are also created.

Access the Grafana Web UI at: `https://<k8sClusterIP>/grafana/login`. The default login credentials are user `admin` and password `prom-operator`.

RELATED DOCUMENTATION

vRouter Session Analytics in Contrail Networking | 179
Centralized Logging | 187

Contrail Networking Metric List

Table 11: Contrail Networking Metric List

Metric Name	Type	Description
controller_state	gauge	Controller state (0=Functional, 1=Non-Functional).
controller_connection_status	gauge	Connection status (0=Up, 1=Down, 2=Initializing).
controller_bgp_router_output_queue_depth	gauge	BGP router output queue depth.
controller_bgp_router_num_bgp_peers	gauge	Number of BGP peers.
controller_bgp_router_num_up_bgp_peers	gauge	Number of up BGP peers.
controller_bgp_router_num_deleting_bgp_peers	gauge	Number of deleting BGP peers.
controller_bgp_router_num_bgpaas_peers	gauge	Number of BGPaas peers.
controller_bgp_router_num_up_bgpaas_peers	gauge	Number of up BGPaas peers.
controller_bgp_router_num_deleting_bgpaas_peers	gauge	Number of deleting BGPaas peers.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
controller_bgp_router_num_xmpp_peers	gauge	Number of XMPP peers.
controller_bgp_router_num_up_xmpp_peers	gauge	Number of up XMPP peers.
controller_bgp_router_num_deleting_xmpp_peers	gauge	Number of deleting XMPP peers.
controller_bgp_router_num_routing_instances	gauge	BGP router number of routing instances.
controller_bgp_router_num_deleting_routing_instances	gauge	BGP router number of deleting routing instances.
controller_bgp_router_num_service_chains	gauge	Number of service chains.
controller_bgp_router_num_down_service_chains	gauge	Number of down service chains.
controller_bgp_router_num_static_routes	gauge	Number of static routes.
controller_bgp_router_num_down_static_routes	gauge	Number of down static routes.
controller_bgp_router_ifmap_num_peer_clients	gauge	Number of IF-MAP peer clients.
controller_bgp_router_config_db_conn_status	gauge	Status of config database connection (0=Down, 1=Up).
controller_bgp_peer_state	gauge	BGP peer state (0=Idle, 1=Active, 2=Connect, 3=OpenSent, 4=OpenConfirm, 5=Established).

Table 11: Contrail Networking Metric List ***(Continued)***

Metric Name	Type	Description
controller_bgp_peer_flaps_total	counter	BGP peer total flaps.
controller_bgp_peer_received_messages_total	counter	Total BGP peer messages received.
controller_bgp_peer_received_open_messages_total	counter	Total BGP peer open messages received.
controller_bgp_peer_received_keepalive_messages_total	counter	Total BGP peer keepalive messages received.
controller_bgp_peer_received_notification_messages_total	counter	Total BGP peer notification messages received.
controller_bgp_peer_received_update_messages_total	counter	Total BGP peer update messages received.
controller_bgp_peer_received_close_messages_total	counter	Total BGP peer close messages received.
controller_bgp_peer_sent_messages_total	counter	Total BGP peer messages sent.
controller_bgp_peer_sent_open_messages_total	counter	Total BGP peer open messages sent.
controller_bgp_peer_sent_keepalive_messages_total	counter	Total BGP peer keepalive messages sent.
controller_bgp_peer_sent_notification_messages_total	counter	Total BGP peer notification messages sent.
controller_bgp_peer_sent_update_messages_total	counter	Total BGP peer update messages sent.
controller_bgp_peer_sent_close_messages_total	counter	Total BGP peer close messages sent.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
controller_bgp_peer_received_reachable_routes_total	counter	Total BGP peer reachable routes received.
controller_bgp_peer_received_unreachable_routes_total	counter	Total BGP peer unreachable routes received.
controller_bgp_peer_received_end_of_rib_total	counter	Total BGP peer end-of-RIB markers received.
controller_bgp_peer_sent_reachable_routes_total	counter	Total BGP peer reachable routes sent.
controller_bgp_peer_sent_unreachable_routes_total	counter	Total BGP peer unreachable routes sent.
controller_bgp_peer_sent_end_of_rib_total	counter	Total BGP peer end-of-RIB markers sent.
controller_bgp_peer_received_bytes_total	counter	Total BGP peer bytes received.
controller_bgp_peer_receive_socket_calls_total	counter	Total BGP peer receive socket calls.
controller_bgp_peer_blocked_receive_socket_calls_microsecond_duration_total	counter	BGP peer total microseconds blocked on socket receive calls.
controller_bgp_peer_blocked_receive_socket_calls_total	counter	Total BGP peer receive socket calls blocked.
controller_bgp_peer_sent_bytes_total	counter	Total BGP peer bytes sent.
controller_bgp_peer_send_socket_calls_total	counter	Total BGP peer send socket calls.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
controller_bgp_peer_blocked_send_socket_calls_microsecond_duration_total	counter	BGP peer total microseconds blocked on socket send calls.
controller_bgp_peer_blocked_send_socket_calls_total	counter	Total BGP peer send socket calls blocked.
controller_bgp_peer_route_update_error_bad_inet6_xml_token_total	counter	BGP peer total route update errors (bad inet6 XML token).
controller_bgp_peer_route_update_error_bad_inet6_prefix_total	counter	BGP peer total route update errors (bad inet6 prefix).
controller_bgp_peer_route_update_error_bad_inet6_nexthop_total	counter	BGP peer total route update errors (bad inet6 next hop).
controller_bgp_peer_route_update_error_bad_inet6_afi_safi_total	counter	BGP peer total route update errors (bad inet6 AFI/SAFI).
controller_bgp_peer_received_route_paths_total	counter	Total BGP peer route paths received.
controller_bgp_peer_received_route_primary_paths_total	counter	Total BGP peer route primary paths received.
controller_xmpp_peer_state	counter	XMPP peer state (0=Idle, 1=Active, 2=Connect, 3=OpenSent, 4=OpenConfirm, 5=Established).
controller_xmpp_peer_received_messages_total	counter	Total messages received from XMPP peer.
controller_xmpp_peer_received_open_messages_total	counter	Total open messages received from XMPP peer.
controller_xmpp_peer_received_keepalive_messages_total	counter	Total keepalive messages received from XMPP peer.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
controller_xmpp_peer_received_notification_messages_total	counter	Total notification messages received from XMPP peer.
controller_xmpp_peer_received_update_messages_total	counter	Total update messages received from XMPP peer.
controller_xmpp_peer_received_close_messages_total	counter	Total close messages received from XMPP peer.
controller_xmpp_peer_sent_messages_total	counter	Total messages sent to XMPP peer.
controller_xmpp_peer_sent_open_messages_total	counter	Total open messages sent to XMPP peer.
controller_xmpp_peer_sent_keepalive_messages_total	counter	Total keepalive messages sent to XMPP peer.
controller_xmpp_peer_sent_notification_messages_total	counter	Total notification messages sent to XMPP peer.
controller_xmpp_peer_sent_update_messages_total	counter	Total update messages sent to XMPP peer.
controller_xmpp_peer_sent_close_messages_total	counter	Total close messages sent to XMPP peer.
controller_xmpp_peer_received_reachable_routes_total	counter	Total reachable routes received from XMPP peer.
controller_xmpp_peer_received_unreachable_routes_total	counter	Total unreachable routes received from XMPP peer.
controller_xmpp_peer_received_end_of_rib_total	counter	Total end-of-RIB markers received from XMPP peer.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
controller_xmpp_peer_sent_reachable_routes_total	counter	Total reachable routes sent to XMPP peer.
controller_xmpp_peer_sent_unreachable_routes_total	counter	Total unreachable routes sent to XMPP peer.
controller_xmpp_peer_sent_end_of_rib_total	counter	Total end-of-RIB markers sent to XMPP peer.
controller_xmpp_peer_route_update_error_bad_inet6_xml_token_total	counter	XMPP peer total route update errors (bad inet6 XML token).
controller_xmpp_peer_route_update_error_bad_inet6_prefix_total	counter	XMPP peer total route update errors (bad inet6 prefix).
controller_xmpp_peer_route_update_error_bad_inet6_nexthop_total	counter	XMPP peer total route update errors (bad inet6 next hop).
controller_xmpp_peer_route_update_error_bad_inet6_afi_safi_total	counter	XMPP peer total route update errors (bad inet6 AFI/SAFI).
controller_xmpp_peer_received_route_paths_total	counter	Total XMPP peer route paths received.
controller_xmpp_peer_received_route_primary_paths_total	counter	Total XMPP peer route primary paths received.
controller_peer_received_reachable_routes_total	counter	Total reachable routes received from peer.
controller_peer_received_unreachable_routes_total	counter	Total unreachable routes received from peer.
controller_peer_received_end_of_rib_total	counter	Total end-of-RIB markers received from peer.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
controller_peer_sent_reachable_routes_total	counter	Total reachable routes sent to peer.
controller_peer_sent_unreachable_routes_total	counter	Total unreachable routes sent to peer.
controller_peer_sent_end_of_rib_total	counter	Total end-of-RIB markers sent to peer.
controller_virtual_network_routing_instance_ipv4_table_prefixes	gauge	Virtual network IPv4 routing table prefixes.
controller_virtual_network_routing_instance_ipv4_table_primary_paths	gauge	Virtual network IPv4 routing table primary paths.
controller_virtual_network_routing_instance_ipv4_table_secondary_paths	gauge	Virtual network IPv4 routing table secondary paths.
controller_virtual_network_routing_instance_ipv4_table_infeasible_paths	gauge	Virtual network IPv4 routing table infeasible paths.
controller_virtual_network_routing_instance_ipv4_table_total_paths	gauge	Virtual network IPv4 routing table total paths.
controller_virtual_network_routing_instance_ipv6_table_prefixes	gauge	Virtual network IPv6 routing table prefixes.
controller_virtual_network_routing_instance_ipv6_table_primary_paths	gauge	Virtual network IPv6 routing table primary paths.
controller_virtual_network_routing_instance_ipv6_table_secondary_paths	gauge	Virtual network IPv6 routing table secondary paths.
controller_virtual_network_routing_instance_ipv6_table_infeasible_paths	gauge	Virtual network IPv6 routing table infeasible paths.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
controller_virtual_network_routing_instance_ipv6_table_total_paths	gauge	Virtual network IPv6 routing table total paths.
controller_virtual_network_routing_instance_evpn_table_prefixes	gauge	Virtual network EVPN routing table prefixes.
controller_virtual_network_routing_instance_evpn_table_primary_paths	gauge	Virtual network EVPN routing table primary paths.
controller_virtual_network_routing_instance_evpn_table_secondary_paths	gauge	Virtual network EVPN routing table secondary paths.
controller_virtual_network_routing_instance_evpn_table_infeasible_paths	gauge	Virtual network EVPN routing table infeasible paths.
controller_virtual_network_routing_instance_evpn_table_total_paths	gauge	Virtual network EVPN routing table total paths.
controller_virtual_network_routing_instance_ermvpn_table_prefixes	gauge	Virtual network ERMVPN routing table prefixes.
controller_virtual_network_routing_instance_ermvpn_table_primary_paths	gauge	Virtual network ERMVPN routing table primary paths.
controller_virtual_network_routing_instance_ermvpn_table_secondary_paths	gauge	Virtual network ERMVPN routing table secondary paths.
controller_virtual_network_routing_instance_ermvpn_table_infeasible_paths	gauge	Virtual network ERMVPN routing table infeasible paths.
controller_virtual_network_routing_instance_ermvpn_table_total_paths	gauge	Virtual network ERMVPN routing table total paths.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
controller_virtual_network_routing_instance_mvpn_table_prefixes	gauge	Virtual network MVPN routing table prefixes.
controller_virtual_network_routing_instance_mvpn_table_primary_paths	gauge	Virtual network MVPN routing table primary paths.
controller_virtual_network_routing_instance_mvpn_table_secondary_paths	gauge	Virtual network MVPN routing table secondary paths.
controller_virtual_network_routing_instance_mvpn_table_infeasible_paths	gauge	Virtual network MVPN routing table infeasible paths.
controller_virtual_network_routing_instance_mvpn_table_total_paths	gauge	Virtual network MVPN routing table total paths.
virtual_router_cpu_1min_load_avg	gauge	Virtual router CPU 1 minute load average.
virtual_router_cpu_5min_load_avg	gauge	Virtual router CPU 5 minute load average.
virtual_router_cpu_15min_load_avg	gauge	Virtual router CPU 15 minute load average.
virtual_router_system_memory_bytes	gauge	Virtual router total system memory.
virtual_router_system_memory_free_bytes	gauge	Virtual router system memory free.
virtual_router_system_memory_used_bytes	gauge	Virtual router system memory used.
virtual_router_system_memory_cached_bytes	gauge	Virtual router system memory cached.
virtual_router_system_memory_buffers	gauge	Virtual router system memory buffers.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
virtual_router_virtual_memory_kilobytes	gauge	Virtual router virtual memory.
virtual_router_resident_memory_kilobytes	gauge	Virtual router resident memory.
virtual_router_peak_virtual_memory_bytes	gauge	Virtual router peak virtual memory.
virtual_router_phys_if_input_packets_total	counter	Total packets received by physical interface.
virtual_router_phys_if_output_packets_total	counter	Total packets sent by physical interface.
virtual_router_phys_if_input_bytes_total	counter	Total bytes received by physical interface.
virtual_router_phys_if_output_bytes_total	counter	Total bytes sent by physical interface.
virtual_router_input_packets_total	counter	Total packets received by virtual router.
virtual_router_output_packets_total	counter	Total packets sent by virtual router.
virtual_router_input_bytes_total	counter	Total bytes received by virtual router.
virtual_router_output_bytes_total	counter	Total bytes sent by virtual router.
virtual_router_flows_total	counter	Total virtual router flows.
virtual_router_aged_flows_total	counter	Total virtual router aged flows.
virtual_router_active_flows	gauge	Current virtual router active flows.
virtual_router_hold_flows	gauge	Current virtual router hold flows.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
virtual_router_added_flows_diff_total	gauge	Virtual router added flows since last sample.
virtual_router_exception_packets_total	counter	Total virtual router exception packets.
virtual_router_exception_packets_allowed_total	counter	Total virtual router exception packets allowed.
virtual_router_exception_packets_dropped_total	counter	Total virtual router exception packets dropped.
virtual_router_dropped_packets_total	counter	Total packets dropped.
virtual_router_vhost_dropped_packets_total	counter	Total virtual host packets dropped.
virtual_router_input_bandwidth_utilization	gauge	Ingress bandwidth of physical interface where the value is obtained by dividing the bandwidth computed in bps by speed of the physical interface.
virtual_router_output_bandwidth_utilization	gauge	Egress bandwidth of physical interface where the value is obtained by dividing the bandwidth computed in bps by speed of the physical interface.
virtual_router_vhost_interface_input_bytes_total	counter	Total bytes received by virtual host interface.
virtual_router_vhost_interface_output_bytes_total	counter	Total bytes sent by virtual host interface.
virtual_router_vhost_interface_input_packets_total	counter	Total packets received by virtual host interface.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
virtual_router_vhost_interface_output_packets_total	counter	Total packets sent by virtual host interface.
virtual_router_virtual_networks	gauge	Current number of virtual networks.
virtual_router_virtual_machines	gauge	Current number of virtual machines.
virtual_router_virtual_machine_interfaces	gauge	Current number of virtual machine interfaces.
virtual_router_interfaces_down	gauge	Current number of down interfaces.
virtual_router_agent_state	gauge	Virtual router agent state (0=Functional, 1=Non-Functional).
virtual_router_connection_status	gauge	Connection status (0=Up, 1=Down, 2=Initializing).
virtual_router_virtual_network_input_packets_total	counter	Total input packets received.
virtual_router_virtual_network_output_packets_total	counter	Total output packets sent.
virtual_router_virtual_network_input_bytes_total	counter	Total input bytes received.
virtual_router_virtual_network_output_bytes_total	counter	Total output bytes sent.
virtual_router_virtual_network_flows	gauge	Current number of flows.
virtual_router_virtual_network_ingress_flows	gauge	Current number of ingress flows.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
virtual_router_virtual_network_egress_flows	gauge	Current number of egress flows.
virtual_router_virtual_network_floating_ips	gauge	Current number of floating IP addresses.
virtual_router_virtual_network_flow_policy_rule_hits_total	counter	Total number of flow policy rule hits.
virtual_router_virtual_network_vrf_bridge_route_table_entries	gauge	Virtual routing and forwarding bridge route table current entries.
virtual_router_virtual_network_vrf_evpn_route_table_entries	gauge	Virtual routing and forwarding EVPN route table current entries.
virtual_router_virtual_network_vrf_inet4_unicast_route_table_entries	gauge	Virtual routing and forwarding inet4 unicast table current entries.
virtual_router_virtual_network_vrf_inet4_multicast_route_table_entries	gauge	Virtual routing and forwarding inet4 multicast table current entries.
virtual_router_virtual_network_vrf_inet6_unicast_route_table_entries	gauge	Virtual routing and forwarding inet6 unicast table current entries.
virtual_router_virtual_machine_interface_input_bytes_total	counter	Total input bytes received by virtual machine interface.
virtual_router_virtual_machine_interface_output_bytes_total	counter	Total output bytes sent by virtual machine interface.
virtual_router_virtual_machine_interface_input_packets_total	counter	Total input packets received by virtual machine interface.
virtual_router_virtual_machine_interface_output_packets_total	counter	Total output packets sent by virtual machine interface.

Table 11: Contrail Networking Metric List *(Continued)*

Metric Name	Type	Description
virtual_router_virtual_machine_interface_active_flows	gauge	Current virtual machine interface active flows.
virtual_router_virtual_machine_interface_hold_flows	gauge	Current virtual machine interface hold flows.
virtual_router_virtual_machine_interface_added_flows_diff_total	gauge	Virtual machine interface added flows since last sample.
virtual_router_virtual_machine_interface_dropped_packets_total	counter	Virtual machine interface total dropped packets.

RELATED DOCUMENTATION

Contrail Networking Analytics | 93
Kubernetes Metric List | 113
Cluster Node Metric List | 152
Contrail Networking Alert List | 169

Kubernetes Metric List

Table 12: Kubernetes Metric List

Metric Name	Type	Description
apiextensions_openapi_v2_regeneration_count	counter	[ALPHA] Counter of OpenAPI v2 spec regeneration count broken down by causing CRD name and reason.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
apiserver_admission_controller_admission_duration_seconds	histogram	[ALPHA] Admission controller latency histogram in seconds, identified by name and broken out for each operation and API resource and type (validate or admit).
apiserver_admission_step_admission_duration_seconds_summary	summary	[ALPHA] Admission sub-step latency summary in seconds, broken out for each operation and API resource and step type (validate or admit).
apiserver_admission_webhook_admission_duration_seconds	histogram	[ALPHA] Admission webhook latency histogram in seconds, identified by name and broken out for each operation and API resource and type (validate or admit).
apiserver_admission_webhook_rejection_count	counter	[ALPHA] Admission webhook rejection count, identified by name and broken out for each admission type (validating or admit) and operation. Additional labels specify an error type (calling_webhook_error or apiserver_internal_error if an error occurred; no_error otherwise) and optionally a non-zero rejection code if the webhook rejects the request with an HTTP status code (honored by the apiserver when the code is greater or equal to 400). Codes greater than 600 are truncated to 600, to keep the metrics cardinality bounded.
apiserver_audit_event_total	counter	[ALPHA] Counter of audit events generated and sent to the audit backend.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
apiserver_audit_requests_rejected_total	counter	[ALPHA] Counter of apiserver requests rejected due to an error in audit logging backend.
apiserver_client_certificate_expiration_seconds	histogram	[ALPHA] Distribution of the remaining lifetime on the certificate used to authenticate a request.
apiserver_current_inflight_requests	gauge	[ALPHA] Maximal number of currently used inflight request limit of this apiserver per request kind in last second.
apiserver_current_inqueue_requests	gauge	[ALPHA] Maximal number of queued requests in this apiserver per request kind in last second.
apiserver_envelope_encryption_dek_cache_fill_percent	gauge	[ALPHA] Percent of the cache slots currently occupied by cached DEKs.
apiserver_flowcontrol_current_executing_requests	gauge	[ALPHA] Number of requests currently executing in the API Priority and Fairness system.
apiserver_flowcontrol_current_inqueue_requests	gauge	[ALPHA] Number of requests currently pending in queues of the API Priority and Fairness system.
apiserver_flowcontrol_dispatched_requests_total	counter	[ALPHA] Number of requests released by API Priority and Fairness system for service.
apiserver_flowcontrol_priority_level_request_count_samples	histogram	[ALPHA] Periodic observations of the number of requests.
apiserver_flowcontrol_priority_level_request_count_watermarks	histogram	[ALPHA] Watermarks of the number of requests.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
apiserver_flowcontrol_read_vs_write_request_count_samples	histogram	[ALPHA] Periodic observations of the number of requests.
apiserver_flowcontrol_read_vs_write_request_count_watermarks	histogram	[ALPHA] Watermarks of the number of requests.
apiserver_flowcontrol_request_concurrency_limit	gauge	[ALPHA] Shared concurrency limit in the API Priority and Fairness system.
apiserver_flowcontrol_request_execution_seconds	histogram	[ALPHA] Duration of request execution in the API Priority and Fairness system.
apiserver_flowcontrol_request_queue_length_after_enqueue	histogram	[ALPHA] Length of queue in the API Priority and Fairness system, as seen by each request after it is enqueued.
apiserver_flowcontrol_request_wait_duration_seconds	histogram	[ALPHA] Length of time a request spent waiting in its queue.
apiserver_init_events_total	counter	[ALPHA] Counter of init events processed in watchcache broken by resource type.
apiserver_longrunning_gauge	gauge	[ALPHA] Gauge of all active long-running apiserver requests broken out by verb, group, version, resource, scope and component. Not all requests are tracked this way.
apiserver_registered_watchers	gauge	[ALPHA] Number of currently registered watchers for a given resources.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
apiserver_request_duration_seconds	histogram	[ALPHA] Response latency distribution in seconds for each verb, dry run value, group, version, resource, subresource, scope and component.
apiserver_request_filter_duration_seconds	histogram	[ALPHA] Request filter latency distribution in seconds, for each filter type.
apiserver_request_total	counter	[ALPHA] Counter of apiserver requests broken out for each verb, dry run value, group, version, resource, scope, component, and HTTP response contentType and code.
apiserver_requested_deprecated_apis	gauge	[ALPHA] Gauge of deprecated APIs that have been requested, broken out by API group, version, resource, subresource, and removed_release.
apiserver_response_sizes	histogram	[ALPHA] Response size distribution in bytes for each group, version, verb, resource, subresource, scope and component.
apiserver_selfrequest_total	counter	[ALPHA] Counter of apiserver self-requests broken out for each verb, API resource and subresource.
apiserver_storage_data_key_generation_duration_seconds	histogram	[ALPHA] Latencies in seconds of data encryption key (DEK) generation operations.
apiserver_storage_data_key_generation_failures_total	counter	[ALPHA] Total number of failed data encryption key (DEK) generation operations.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
apiserver_storage_envelope_transformation_cache_misses_total	counter	[ALPHA] Total number of cache misses while accessing key decryption key (KEK).
apiserver_tls_handshake_errors_total	counter	[ALPHA] Number of requests dropped with 'TLS handshake error from' error.
apiserver_watch_events_sizes	histogram	[ALPHA] Watch event size distribution in bytes.
apiserver_watch_events_total	counter	[ALPHA] Number of events sent in watch clients.
authenticated_user_requests	counter	[ALPHA] Counter of authenticated requests broken out by username.
authentication_attempts	counter	[ALPHA] Counter of authenticated attempts.
authentication_duration_seconds	histogram	[ALPHA] Authentication duration in seconds broken out by result.
authentication_token_cache_active_fetch_count	gauge	[ALPHA]
authentication_token_cache_fetch_total	counter	[ALPHA]
authentication_token_cache_request_duration_seconds	histogram	[ALPHA]
authentication_token_cache_request_total	counter	[ALPHA]

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
cadvisor_version_info	gauge	A metric with a constant '1' value labeled by kernel version, OS version, docker version, cadvisor version and cadvisor revision.
container_cpu_cfs_periods_total	counter	Number of elapsed enforcement period intervals.
container_cpu_cfs_throttled_periods_total	counter	Number of throttled period intervals.
container_cpu_cfs_throttled_seconds_total	counter	Total time duration the container has been throttled.
container_cpu_load_average_10s	gauge	Value of container CPU load average over the last 10 seconds.
container_cpu_system_seconds_total	counter	Cumulative system CPU time consumed in seconds.
container_cpu_usage_seconds_total	counter	Cumulative CPU time consumed in seconds.
container_cpu_user_seconds_total	counter	Cumulative user CPU time consumed in seconds.
container_file_descriptors	gauge	Number of open file descriptors for the container.
container_fs_inodes_free	gauge	Number of available Inodes.
container_fs_inodes_total	gauge	Number of Inodes.
container_fs_io_current	gauge	Number of I/Os currently in progress.
container_fs_io_time_seconds_total	counter	Cumulative count of seconds spent doing I/Os.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
container_fs_io_time_weighted_seconds_total	counter	Cumulative weighted I/O time in seconds.
container_fs_limit_bytes	gauge	Number of bytes that can be consumed by the container on this filesystem.
container_fs_read_seconds_total	counter	Cumulative count of seconds spent reading.
container_fs_reads_bytes_total	counter	Cumulative count of bytes read.
container_fs_reads_merged_total	counter	Cumulative count of reads merged.
container_fs_reads_total	counter	Cumulative count of reads completed.
container_fs_sector_reads_total	counter	Cumulative count of sector reads completed.
container_fs_sector_writes_total	counter	Cumulative count of sector writes completed.
container_fs_usage_bytes	gauge	Number of bytes that are consumed by the container on this filesystem.
container_fs_write_seconds_total	counter	Cumulative count of seconds spent writing.
container_fs_writes_bytes_total	counter	Cumulative count of bytes written.
container_fs_writes_merged_total	counter	Cumulative count of writes merged.
container_fs_writes_total	counter	Cumulative count of writes completed.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
container_last_seen	gauge	Last time a container was seen by the exporter.
container_memory_cache	gauge	Number of bytes of page cache memory.
container_memory_failcnt	counter	Number of memory usage hits limits.
container_memory_failures_total	counter	Cumulative count of memory allocation failures.
container_memory_mapped_file	gauge	Size of memory mapped files in bytes.
container_memory_max_usage_bytes	gauge	Maximum memory usage recorded in bytes.
container_memory_rss	gauge	Size of RSS in bytes.
container_memory_swap	gauge	Container swap usage in bytes.
container_memory_usage_bytes	gauge	Current memory usage in bytes, including all memory regardless of when it was accessed.
container_memory_working_set_bytes	gauge	Current working set in bytes.
container_network_receive_bytes_total	counter	Cumulative count of bytes received.
container_network_receive_errors_total	counter	Cumulative count of errors encountered while receiving.
container_network_receive_packets_dropped_total	counter	Cumulative count of packets dropped while receiving.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
container_network_receive_packets_total	counter	Cumulative count of packets received.
container_network_transmit_bytes_total	counter	Cumulative count of bytes transmitted.
container_network_transmit_errors_total	counter	Cumulative count of errors encountered while transmitting.
container_network_transmit_packets_dropped_total	counter	Cumulative count of packets dropped while transmitting.
container_network_transmit_packets_total	counter	Cumulative count of packets transmitted.
container_processes	gauge	Number of processes running inside the container.
container_scrape_error	gauge	1 if there was an error while getting container metrics, 0 otherwise.
container_sockets	gauge	Number of open sockets for the container.
container_spec_cpu_period	gauge	CPU period of the container.
container_spec_cpu_quota	gauge	CPU quota of the container.
container_spec_cpu_shares	gauge	CPU share of the container.
container_spec_memory_limit_bytes	gauge	Memory limit for the container.
container_spec_memory_reservation_limit_bytes	gauge	Memory reservation limit for the container.
container_spec_memory_swap_limit_bytes	gauge	Memory swap limit for the container.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
container_start_time_seconds	gauge	Start time of the container since Unix epoch in seconds.
container_tasks_state	gauge	Number of tasks in given state.
container_threads	gauge	Number of threads running inside the container.
container_threads_max	gauge	Maximum number of threads allowed inside the container, infinity if value is zero.
container_ulimits_soft	gauge	Soft ulimit values for the container root process. Unlimited if -1, except priority and nice.
coredns_build_info	gauge	A metric with a constant '1' value labeled by version, revision, and goversion from which CoreDNS was built.
coredns_cache_entries	gauge	The number of elements in the cache.
coredns_cache_hits_total	counter	The count of cache hits.
coredns_cache_misses_total	counter	The count of cache misses.
coredns_dns_request_duration_seconds	histogram	Histogram of the time (in seconds) each request took.
coredns_dns_request_size_bytes	histogram	Size of the EDNS0 UDP buffer in bytes (64K for TCP).
coredns_dns_requests_total	counter	Counter of DNS requests made per zone, protocol and family.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
coredns_dns_response_size_bytes	histogram	Size of the returned response in bytes.
coredns_dns_responses_total	counter	Counter of response status codes.
coredns_forward_healthcheck_failures_total	counter	Counter of the number of failed healthchecks.
coredns_forward_max_concurrent_rejects_total	counter	Counter of the number of queries rejected because the concurrent queries were at maximum.
coredns_forward_request_duration_seconds	histogram	Histogram of the time each request took.
coredns_forward_requests_total	counter	Counter of requests made per upstream.
coredns_forward_responses_total	counter	Counter of requests made per upstream.
coredns_health_request_duration_seconds	histogram	Histogram of the time (in seconds) each request took.
coredns_panics_total	counter	A metrics that counts the number of panics.
coredns_plugin_enabled	gauge	A metric that indicates whether a plugin is enabled on per server and zone basis.
etcd_db_total_size_in_bytes	gauge	[ALPHA] Total size of the etcd database file physically allocated in bytes.
etcd_object_counts	gauge	[ALPHA] Number of stored objects at the time of last check split by kind.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
etcd_request_duration_seconds	histogram	[ALPHA] Etcd request latency in seconds for each operation and object type.
kube_certificatesigningrequest_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_certificatesigningrequest_cert_length	gauge	Length of the issued certificate.
kube_certificatesigningrequest_condition	gauge	The number of each certificatesigningrequest condition.
kube_certificatesigningrequest_created	gauge	Unix creation timestamp.
kube_certificatesigningrequest_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_configmap_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_configmap_created	gauge	Unix creation timestamp.
kube_configmap_info	gauge	Information about configmap.
kube_configmap_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_configmap_metadata_resource_version	gauge	Resource version representing a specific version of the configmap.
kube_cronjob_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_cronjob_created	gauge	Unix creation timestamp.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_cronjob_info	gauge	Info about cronjob.
kube_cronjob_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_cronjob_metadata_resource_version	gauge	Resource version representing a specific version of the cronjob.
kube_cronjob_next_schedule_time	gauge	Next time the cronjob should be scheduled. The time after lastScheduleTime, or after the cron job's creation time if it's never been scheduled. Use this to determine if the job is delayed.
kube_cronjob_spec_failed_job_history_limit	gauge	Failed job history limit tells the controller how many failed jobs should be preserved.
kube_cronjob_spec_starting_deadline_seconds	gauge	Deadline in seconds for starting the job if it misses scheduled time for any reason.
kube_cronjob_spec_successful_job_history_limit	gauge	Successful job history limit tells the controller how many completed jobs should be preserved.
kube_cronjob_spec_suspend	gauge	Suspend flag tells the controller to suspend subsequent executions.
kube_cronjob_status_active	gauge	Active holds pointers to currently running jobs.
kube_cronjob_status_last_schedule_time	gauge	LastScheduleTime keeps information of when was the last time the job was successfully scheduled.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_daemonset_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_daemonset_created	gauge	Unix creation timestamp.
kube_daemonset_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_daemonset_metadata_generation	gauge	Sequence number representing a specific generation of the desired state.
kube_daemonset_status_current_number_scheduled	gauge	The number of nodes running at least one daemon pod and are supposed to.
kube_daemonset_status_desired_number_scheduled	gauge	The number of nodes that should be running the daemon pod.
kube_daemonset_status_number_available	gauge	The number of nodes that should be running the daemon pod and have one or more of the daemon pod running and available.
kube_daemonset_status_number_misscheduled	gauge	The number of nodes running a daemon pod but are not supposed to.
kube_daemonset_status_number_ready	gauge	The number of nodes that should be running the daemon pod and have one or more of the daemon pod running and ready.
kube_daemonset_status_number_unavailable	gauge	The number of nodes that should be running the daemon pod and have none of the daemon pod running and available.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_daemonset_status_observed_generation	gauge	The most recent generation observed by the daemon set controller.
kube_daemonset_status_updated_number_scheduled	gauge	The total number of nodes that are running updated daemon pod.
kube_deployment_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_deployment_created	gauge	Unix creation timestamp.
kube_deployment_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_deployment_metadata_generation	gauge	Sequence number representing a specific generation of the desired state.
kube_deployment_spec_paused	gauge	Whether the deployment is paused and will not be processed by the deployment controller.
kube_deployment_spec_replicas	gauge	Number of desired pods for a deployment.
kube_deployment_spec_strategy_rollingupdate_max_surge	gauge	Maximum number of replicas that can be scheduled above the desired number of replicas during a rolling update of a deployment.
kube_deployment_spec_strategy_rollingupdate_max_unavailable	gauge	Maximum number of unavailable replicas during a rolling update of a deployment.
kube_deployment_status_condition	gauge	The current status conditions of a deployment.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_deployment_status_observed_generation	gauge	The generation observed by the deployment controller.
kube_deployment_status_replicas	gauge	The number of replicas per deployment.
kube_deployment_status_replicas_available	gauge	The number of available replicas per deployment.
kube_deployment_status_replicas_ready	gauge	The number of ready replicas per deployment.
kube_deployment_status_replicas_unavailable	gauge	The number of unavailable replicas per deployment.
kube_deployment_status_replicas_updated	gauge	The number of updated replicas per deployment.
kube_endpoint_address_available	gauge	Number of addresses available in endpoint.
kube_endpoint_address_not_ready	gauge	Number of addresses not ready in endpoint.
kube_endpoint_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_endpoint_created	gauge	Unix creation timestamp.
kube_endpoint_info	gauge	Information about endpoint.
kube_endpoint_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_endpoint_ports	gauge	Information about the endpoint ports.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_horizontalpodautoscaler_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_horizontalpodautoscaler_info	gauge	Information about this autoscaler.
kube_horizontalpodautoscaler_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_horizontalpodautoscaler_metadata_generation	gauge	The generation observed by the HorizontalPodAutoscaler controller.
kube_horizontalpodautoscaler_spec_max_replicas	gauge	Upper limit for the number of pods that can be set by the autoscaler; cannot be smaller than MinReplicas.
kube_horizontalpodautoscaler_spec_min_replicas	gauge	Lower limit for the number of pods that can be set by the autoscaler, default 1.
kube_horizontalpodautoscaler_spec_target_metric	gauge	The metric specifications used by this autoscaler when calculating the desired replica count.
kube_horizontalpodautoscaler_status_condition	gauge	The condition of this autoscaler.
kube_horizontalpodautoscaler_status_current_replicas	gauge	Current number of replicas of pods managed by this autoscaler.
kube_horizontalpodautoscaler_status_desired_replicas	gauge	Desired number of replicas of pods managed by this autoscaler.
kube_ingress_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_ingress_created	gauge	Unix creation timestamp.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_ingress_info	gauge	Information about ingress.
kube_ingress_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_ingress_metadata_resource_version	gauge	Resource version representing a specific version of ingress.
kube_ingress_path	gauge	Ingress host, paths and backend service information.
kube_ingress_tls	gauge	Ingress TLS host and secret information.
kube_job_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_job_complete	gauge	The job has completed its execution.
kube_job_created	gauge	Unix creation timestamp.
kube_job_failed	gauge	The job has failed its execution.
kube_job_info	gauge	Information about job.
kube_job_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_job_owner	gauge	Information about the Job's owner.
kube_job_spec_active_deadline_seconds	gauge	The duration in seconds relative to the startTime that the job may be active before the system tries to terminate it.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_job_spec_completions	gauge	The desired number of successfully finished pods the job should be run with.
kube_job_spec_parallelism	gauge	The maximum desired number of pods the job should run at any given time.
kube_job_status_active	gauge	The number of actively running pods.
kube_job_status_completion_time	gauge	CompletionTime represents time when the job was completed.
kube_job_status_failed	gauge	The number of pods which reached Phase Failed and the reason for failure.
kube_job_status_start_time	gauge	StartTime represents time when the job was acknowledged by the Job Manager.
kube_job_status_succeeded	gauge	The number of pods which reached Phase Succeeded.
kube_limitrange	gauge	Information about limit range.
kube_limitrange_created	gauge	Unix creation timestamp.
kube_mutatingwebhookconfiguration_created	gauge	Unix creation timestamp.
kube_mutatingwebhookconfiguration_info	gauge	Information about the MutatingWebhookConfiguration.
kube_mutatingwebhookconfiguration_metadata_resource_version	gauge	Resource version representing a specific version of the MutatingWebhookConfiguration.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_namespace_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_namespace_created	gauge	Unix creation timestamp.
kube_namespace_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_namespace_status_condition	gauge	The condition of a namespace.
kube_namespace_status_phase	gauge	Kubernetes namespace status phase.
kube_networkpolicy_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_networkpolicy_created	gauge	Unix creation timestamp of network policy.
kube_networkpolicy_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_networkpolicy_spec_egress_rules	gauge	Number of egress rules on the networkpolicy.
kube_networkpolicy_spec_ingress_rules	gauge	Number of ingress rules on the networkpolicy.
kube_node_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_node_created	gauge	Unix creation timestamp.
kube_node_info	gauge	Information about a cluster node.
kube_node_labels	gauge	Kubernetes labels converted to Prometheus labels.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_node_role	gauge	The role of a cluster node.
kube_node_spec_taint	gauge	The taint of a cluster node.
kube_node_spec_unschedulable	gauge	Whether a node can schedule new pods.
kube_node_status_allocatable	gauge	The allocatable for different resources of a node that are available for scheduling.
kube_node_status_capacity	gauge	The capacity for different resources of a node.
kube_node_status_condition	gauge	The condition of a cluster node.
kube_persistentvolume_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_persistentvolume_capacity_bytes	gauge	Persistentvolume capacity in bytes.
kube_persistentvolume_claim_ref	gauge	Information about the Persistent Volume Claim Reference.
kube_persistentvolume_info	gauge	Information about persistentvolume.
kube_persistentvolume_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_persistentvolume_status_phase	gauge	The phase indicates if a volume is available, bound to a claim, or released by a claim.
kube_persistentvolumeclaim_access_mode	gauge	The access mode(s) specified by the persistent volume claim.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_persistentvolumeclaim_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_persistentvolumeclaim_info	gauge	Information about persistent volume claim.
kube_persistentvolumeclaim_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_persistentvolumeclaim_resource_requests_storage_bytes	gauge	The capacity of storage requested by the persistent volume claim.
kube_persistentvolumeclaim_status_condition	gauge	Information about status of different conditions of persistent volume claim.
kube_persistentvolumeclaim_status_phase	gauge	The phase the persistent volume claim is currently in.
kube_pod_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_pod_completion_time	gauge	Completion time in Unix timestamp for a pod.
kube_pod_container_info	gauge	Information about a container in a pod.
kube_pod_container_resource_limits	gauge	The number of requested limit resource by a container.
kube_pod_container_resource_requests	gauge	The number of requested request resource by a container.
kube_pod_container_state_started	gauge	Start time in Unix timestamp for a pod container.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_pod_container_status_last_terminated_reason	gauge	Describes the last reason the container was in terminated state.
kube_pod_container_status_ready	gauge	Describes whether the containers readiness check succeeded.
kube_pod_container_status_restarts_total	counter	The number of container restarts per container.
kube_pod_container_status_running	gauge	Describes whether the container is currently in running state.
kube_pod_container_status_terminated	gauge	Describes whether the container is currently in terminated state.
kube_pod_container_status_terminated_reason	gauge	Describes the reason the container is currently in terminated state.
kube_pod_container_status_waiting	gauge	Describes whether the container is currently in waiting state.
kube_pod_container_status_waiting_reason	gauge	Describes the reason the container is currently in waiting state.
kube_pod_created	gauge	Unix creation timestamp.
kube_pod_deletion_timestamp	gauge	Unix deletion timestamp.
kube_pod_info	gauge	Information about pod.
kube_pod_init_container_info	gauge	Information about an init container in a pod.
kube_pod_init_container_resource_limits	gauge	The number of requested limit resource by an init container.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_pod_init_container_resource_requests	gauge	The number of requested request resource by an init container.
kube_pod_init_container_status_last_terminated_reason	gauge	Describes the last reason the init container was in terminated state.
kube_pod_init_container_status_ready	gauge	Describes whether the init containers readiness check succeeded.
kube_pod_init_container_status_restarts_total	counter	The number of restarts for the init container.
kube_pod_init_container_status_running	gauge	Describes whether the init container is currently in running state.
kube_pod_init_container_status_terminated	gauge	Describes whether the init container is currently in terminated state.
kube_pod_init_container_status_terminated_reason	gauge	Describes the last reason the init container was in terminated state.
kube_pod_init_container_status_ready	gauge	Describes whether the init containers readiness check succeeded.
kube_pod_init_container_status_restarts_total	counter	The number of restarts for the init container.
kube_pod_init_container_status_running	gauge	Describes whether the init container is currently in running state.
kube_pod_init_container_status_terminated	gauge	Describes whether the init container is currently in terminated state.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_pod_init_container_status_terminated_reason	gauge	Describes the reason the init container is currently in terminated state.
kube_pod_init_container_status_waiting	gauge	Describes whether the init container is currently in waiting state.
kube_pod_init_container_status_waiting_reason	gauge	Describes the reason the init container is currently in waiting state.
kube_pod_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_pod_overhead_cpu_cores	gauge	The pod overhead in regards to CPU cores associated with running a pod.
kube_pod_overhead_memory_bytes	gauge	The pod overhead in regards to memory associated with running a pod.
kube_pod_owner	gauge	Information about the pod's owner.
kube_pod_restart_policy	gauge	Describes the restart policy in use by this pod.
kube_pod_runtimeclass_name_info	gauge	The runtimeclass associated with the pod.
kube_pod_spec_volumes_persistentvolumeclaims_info	gauge	Information about persistentvolumeclaim volumes in a pod.
kube_pod_spec_volumes_persistentvolumeclaims_readonly	gauge	Describes whether a persistentvolumeclaim is mounted read only.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_pod_start_time	gauge	Start time in Unix timestamp for a pod.
kube_pod_status_phase	gauge	The pods current phase.
kube_pod_status_ready	gauge	Describes whether the pod is ready to serve requests.
kube_pod_status_reason	gauge	The pod status reasons.
kube_pod_status_scheduled	gauge	Describes the status of the scheduling process for the pod.
kube_pod_status_scheduled_time	gauge	Unix timestamp when pod moved into scheduled status.
kube_pod_status_unschedulable	gauge	Describes the unschedulable status for the pod.
kube_poddisruptionbudget_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_poddisruptionbudget_created	gauge	Unix creation timestamp
kube_poddisruptionbudget_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_poddisruptionbudget_status_current_healthy	gauge	Current number of healthy pods.
kube_poddisruptionbudget_status_desired_healthy	gauge	Minimum desired number of healthy pods.
kube_poddisruptionbudget_status_expected_pods	gauge	Total number of pods counted by this disruption budget.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_poddisruptionbudget_status_observed_generation	gauge	Most recent generation observed when updating this PDB status.
kube_poddisruptionbudget_status_pod_disruptions_allowed	gauge	Number of pod disruptions that are currently allowed.
kube_replicaset_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_replicaset_created	gauge	Unix creation timestamp.
kube_replicaset_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_replicaset_metadata_generation	gauge	Sequence number representing a specific generation of the desired state.
kube_replicaset_owner	gauge	Information about the ReplicaSet's owner.
kube_replicaset_spec_replicas	gauge	Number of desired pods for a ReplicaSet.
kube_replicaset_status_fully_labeled_replicas	gauge	The number of fully labeled replicas per ReplicaSet.
kube_replicaset_status_observed_generation	gauge	The generation observed by the ReplicaSet controller.
kube_replicaset_status_ready_replicas	gauge	The number of ready replicas per ReplicaSet.
kube_replicaset_status_replicas	gauge	The number of replicas per ReplicaSet.
kube_replicationcontroller_created	gauge	Unix creation timestamp.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_replicationcontroller_metadata_generation	gauge	Sequence number representing a specific generation of the desired state.
kube_replicationcontroller_owner	gauge	Information about the ReplicationController's owner.
kube_replicationcontroller_spec_replicas	gauge	Number of desired pods for a ReplicationController.
kube_replicationcontroller_status_available_replicas	gauge	The number of available replicas per ReplicationController.
kube_replicationcontroller_status_fully_labeled_replicas	gauge	The number of fully labeled replicas per ReplicationController.
kube_replicationcontroller_status_observed_generation	gauge	The generation observed by the ReplicationController controller.
kube_replicationcontroller_status_ready_replicas	gauge	The number of ready replicas per ReplicationController.
kube_replicationcontroller_status_replicas	gauge	The number of replicas per ReplicationController.
kube_resourcequota	gauge	Information about resource quota.
kube_resourcequota_created	gauge	Unix creation timestamp.
kube_secret_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_secret_created	gauge	Unix creation timestamp.
kube_secret_info	gauge	Information about secret.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_secret_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_secret_metadata_resource_version	gauge	Resource version representing a specific version of secret.
kube_secret_type	gauge	Type about secret.
kube_service_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_service_created	gauge	Unix creation timestamp.
kube_service_info	gauge	Information about service.
kube_service_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_service_spec_external_ip	gauge	Service external ips. One series for each IP.
kube_service_spec_type	gauge	Type about service.
kube_service_status_load_balancer_ingress	gauge	Service load balancer ingress status.
kube_statefulset_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_statefulset_created	gauge	Unix creation timestamp.
kube_statefulset_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_statefulset_metadata_generation	gauge	Sequence number representing a specific generation of the desired state for the StatefulSet.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_statefulset_replicas	gauge	Number of desired pods for a StatefulSet.
kube_statefulset_status_current_revision	gauge	Indicates the version of the StatefulSet used to generate Pods in the sequence [0,currentReplicas).
kube_statefulset_status_observed_generation	gauge	The generation observed by the StatefulSet controller.
kube_statefulset_status_replicas	gauge	The number of replicas per StatefulSet.
kube_statefulset_status_replicas_available	gauge	The number of available replicas per StatefulSet.
kube_statefulset_status_replicas_current	gauge	The number of current replicas per StatefulSet.
kube_statefulset_status_replicas_ready	gauge	The number of ready replicas per StatefulSet.
kube_statefulset_status_replicas_updated	gauge	The number of updated replicas per StatefulSet.
kube_statefulset_status_update_revision	gauge	Indicates the version of the StatefulSet used to generate Pods in the sequence (replicas-updatedReplicas,replicas).
kube_storageclass_annotations	gauge	Kubernetes annotations converted to Prometheus labels.
kube_storageclass_created	gauge	Unix creation timestamp.
kube_storageclass_info	gauge	Information about storageclass.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kube_storageclass_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_validatingwebhookconfiguration_created	gauge	Unix creation timestamp.
kube_validatingwebhookconfiguration_info	gauge	Information about the ValidatingWebhookConfiguration.
kube_validatingwebhookconfiguration_metadata_resource_version	gauge	Resource version representing a specific version of the ValidatingWebhookConfiguration.
kube_volumeattachment_created	gauge	Unix creation timestamp.
kube_volumeattachment_info	gauge	Information about volumeattachment.
kube_volumeattachment_labels	gauge	Kubernetes labels converted to Prometheus labels.
kube_volumeattachment_spec_source_persistentvolume	gauge	PersistentVolume source reference.
kube_volumeattachment_status_attached	gauge	Information about volumeattachment.
kube_volumeattachment_status_attachment_metadata	gauge	The volumeattachment metadata.
kubelet_certificate_manager_client_expiration_renew_errors	counter	[ALPHA] Counter of certificate renewal errors.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kubelet_certificate_manager_client_ttl_seconds	gauge	[ALPHA] Gauge of the TTL (time-to-live) of the Kubelet's client certificate. The value is in seconds until certificate expiry (negative if already expired). If client certificate is invalid or unused, the value will be +INF.
kubelet_cgroup_manager_duration_seconds	histogram	[ALPHA] Duration in seconds for cgroup manager operations. Broken down by method.
kubelet_container_log_filesystem_used_bytes	gauge	[ALPHA] Bytes used by the container's logs on the filesystem.
kubelet_containers_per_pod_count	histogram	[ALPHA] The number of containers per pod.
kubelet_docker_operations_duration_seconds	histogram	[ALPHA] Latency in seconds of Docker operations. Broken down by operation type.
kubelet_docker_operations_errors_total	counter	[ALPHA] Cumulative number of Docker operation errors by operation type.
kubelet_docker_operations_total	counter	[ALPHA] Cumulative number of Docker operations by operation type.
kubelet_http_inflight_requests	gauge	[ALPHA] Number of the inflight http requests.
kubelet_http_requests_duration_seconds	histogram	[ALPHA] Duration in seconds to serve http requests.
kubelet_http_requests_total	counter	[ALPHA] Number of the http requests received since the server started.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kubelet_network_plugin_operations_duration_seconds	histogram	[ALPHA] Latency in seconds of network plugin operations. Broken down by operation type.
kubelet_network_plugin_operations_total	counter	[ALPHA] Cumulative number of network plugin operations by operation type.
kubelet_node_config_error	gauge	[ALPHA] This metric is true (1) if the node is experiencing a configuration-related error, false (0) otherwise.
kubelet_node_name	gauge	[ALPHA] The node's name. The count is always 1.
kubelet_pleg_discard_events	counter	[ALPHA] The number of discard events in PLEG.
kubelet_pleg_last_seen_seconds	gauge	[ALPHA] Timestamp in seconds when PLEG was last seen active.
kubelet_pleg_relist_duration_seconds	histogram	[ALPHA] Duration in seconds for relisting pods in PLEG.
kubelet_pleg_relist_interval_seconds	histogram	[ALPHA] Interval in seconds between relisting in PLEG.
kubelet_pod_start_duration_seconds	histogram	[ALPHA] Duration in seconds for a single pod to go from pending to running.
kubelet_pod_worker_duration_seconds	histogram	[ALPHA] Duration in seconds to sync a single pod. Broken down by operation type: create, update, or sync.
kubelet_pod_worker_start_duration_seconds	histogram	[ALPHA] Duration in seconds from seeing a pod to starting a worker.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kubelet_run_podsandbox_duration_seconds	histogram	[ALPHA] Duration in seconds of the run_podsandbox operations. Broken down by RuntimeClass.Handler.
kubelet_running_containers	gauge	[ALPHA] Number of containers currently running.
kubelet_running_pods	gauge	[ALPHA] Number of pods currently running.
kubelet_runtime_operations_duration_seconds	histogram	[ALPHA] Duration in seconds of runtime operations. Broken down by operation type.
kubelet_runtime_operations_errors_total	counter	[ALPHA] Cumulative number of runtime operation errors by operation type.
kubelet_runtime_operations_total	counter	[ALPHA] Cumulative number of runtime operations by operation type.
kubelet_volume_stats_available_bytes	gauge	[ALPHA] Number of available bytes in the volume.
kubelet_volume_stats_capacity_bytes	gauge	[ALPHA] Capacity in bytes of the volume.
kubelet_volume_stats_inodes	gauge	[ALPHA] Maximum number of inodes in the volume.
kubelet_volume_stats_inodes_free	gauge	[ALPHA] Number of free inodes in the volume.
kubelet_volume_stats_inodes_used	gauge	[ALPHA] Number of used inodes in the volume.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
kubelet_volume_stats_used_bytes	gauge	[ALPHA] Number of used bytes in the volume.
kubeproxy_network_programming_duration_seconds	histogram	[ALPHA] In Cluster Network Programming Latency in seconds.
kubeproxy_sync_proxy_rules_duration_seconds	histogram	[ALPHA] SyncProxyRules latency in seconds.
kubeproxy_sync_proxy_rules_endpoint_changes_pending	gauge	[ALPHA] Pending proxy rules Endpoint changes.
kubeproxy_sync_proxy_rules_endpoint_changes_total	counter	[ALPHA] Cumulative proxy rules Endpoint changes.
kubeproxy_sync_proxy_rules_iptables_restore_failures_total	counter	[ALPHA] Cumulative proxy iptables restore failures.
kubeproxy_sync_proxy_rules_last_queued_timestamp_seconds	gauge	[ALPHA] The last time a sync of proxy rules was queued.
kubeproxy_sync_proxy_rules_last_timestamp_seconds	gauge	[ALPHA] The last time proxy rules were successfully synced.
kubeproxy_sync_proxy_rules_service_changes_pending	gauge	[ALPHA] Pending proxy rules Service changes.
kubeproxy_sync_proxy_rules_service_changes_total	counter	[ALPHA] Cumulative proxy rules Service changes.
kubernetes_build_info	gauge	[ALPHA] A metric with a constant '1' value labeled by major, minor, git version, git commit, git tree state, build date, Go version, and compiler from which Kubernetes was built, and platform on which it is running.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
prober_probe_total	counter	[ALPHA] Cumulative number of a liveness, readiness or startup probe for a container by result.
process_cpu_seconds_total	counter	Total user and system CPU time spent in seconds.
process_max_fds	gauge	Maximum number of open file descriptors.
process_open_fds	gauge	Number of open file descriptors.
process_resident_memory_bytes	gauge	Resident memory size in bytes.
process_start_time_seconds	gauge	Start time of the process since Unix epoch in seconds.
process_virtual_memory_bytes	gauge	Virtual memory size in bytes.
process_virtual_memory_max_bytes	gauge	Maximum amount of virtual memory available in bytes.
rest_client_exec_plugin_certificate_rotation_age	histogram	[ALPHA] Histogram of the number of seconds the last auth exec plugin client certificate lived before being rotated. If auth exec plugin client certificates are unused, histogram will contain no data.
rest_client_exec_plugin_ttl_seconds	gauge	[ALPHA] Gauge of the shortest TTL (time-to-live) of the client certificate(s) managed by the auth exec plugin. The value is in seconds until certificate expiry (negative if already expired). If auth exec plugins are unused or manage no TLS certificates, the value will be +INF.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
rest_client_request_duration_seconds	histogram	[ALPHA] Request latency in seconds. Broken down by verb and URL.
rest_client_requests_total	counter	[ALPHA] Number of HTTP requests, partitioned by status code, method, and host.
serviceaccount_legacy_tokens_total	counter	[ALPHA] Cumulative legacy service account tokens used.
serviceaccount_stale_tokens_total	counter	[ALPHA] Cumulative stale projected service account tokens used.
serviceaccount_valid_tokens_total	counter	[ALPHA] Cumulative valid projected service account tokens used.
ssh_tunnel_open_count	counter	[ALPHA] Counter of ssh tunnel total open attempts.
ssh_tunnel_open_fail_count	counter	[ALPHA] Counter of ssh tunnel failed open attempts.
storage_operation_duration_seconds	histogram	[ALPHA] Storage operation duration.
storage_operation_errors_total	counter	[ALPHA] Storage operation errors.
storage_operation_status_count	counter	[ALPHA] Storage operation return statuses count.
volume_manager_total_volumes	gauge	[ALPHA] Number of volumes in Volume Manager.
workqueue_adds_total	counter	[ALPHA] Total number of adds handled by workqueue.

Table 12: Kubernetes Metric List *(Continued)*

Metric Name	Type	Description
workqueue_depth	gauge	[ALPHA] Current depth of workqueue.
workqueue_longest_running_processor_seconds	gauge	[ALPHA] How many seconds has the longest running processor for workqueue been running.
workqueue_queue_duration_seconds	histogram	[ALPHA] How long in seconds an item stays in workqueue before being requested.
workqueue_retries_total	counter	[ALPHA] Total number of retries handled by workqueue.
workqueue_unfinished_work_seconds	gauge	[ALPHA] How many seconds of work has done that is in progress and hasn't been observed by work_duration. Large values indicate stuck threads. One can deduce the number of stuck threads by observing the rate at which this increases.
workqueue_work_duration_seconds	histogram	[ALPHA] How long in seconds processing an item from workqueue takes.

RELATED DOCUMENTATION

Contrail Networking Analytics | 93
Contrail Networking Metric List | 99
Cluster Node Metric List | 152
Contrail Networking Alert List | 169

Cluster Node Metric List

Table 13: Cluster Node Metric List

Metric Name	Type	Description
node_arp_entries	gauge	ARP entries by device.
node_authorizer_graph_actions_duration_seconds	histogram	[ALPHA] Histogram of duration of graph actions in node authorizer.
node_boot_time_seconds	gauge	Node boot time, in Unix time.
node_context_switches_total	counter	Total number of context switches.
node_cooling_device_cur_state	gauge	Current throttle state of the cooling device.
node_cooling_device_max_state	gauge	Maximum throttle state of the cooling device.
node_cpu_guest_seconds_total	counter	Seconds the CPUs spent in guests (VMs) for each mode.
node_cpu_seconds_total	counter	Seconds the CPUs spent in each mode.
node_disk_info	gauge	info of **/sys/block/<block_device>**.
node_disk_io_now	gauge	The number of I/Os currently in progress.
node_disk_io_time_seconds_total	counter	Total seconds spent doing I/Os.
node_disk_io_time_weighted_seconds_total	counter	The weighted number of seconds spent doing I/Os.
node_disk_read_bytes_total	counter	The total number of bytes read successfully.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_disk_read_time_seconds_total	counter	The total number of seconds spent by all reads.
node_disk_reads_completed_total	counter	The total number of reads completed successfully.
node_disk_reads_merged_total	counter	The total number of reads merged.
node_disk_write_time_seconds_total	counter	This is the total number of seconds spent by all writes.
node_disk_writes_completed_total	counter	The total number of writes completed successfully.
node_disk_writes_merged_total	counter	The number of writes merged.
node_disk_written_bytes_total	counter	The total number of bytes written successfully.
node_dmi_info	gauge	A metric with a constant '1' value labeled by bios_date, bios_release, bios_vendor, bios_version, board_asset_tag, board_name, board_serial, board_vendor, board_version, chassis_asset_tag, chassis_serial, chassis_vendor, chassis_version, product_family, product_name, product_serial, product_sku, product_uuid, product_version, system_vendor if provided by DMI.
node_entropy_available_bits	gauge	Bits of available entropy.
node_entropy_pool_size_bits	gauge	Bits of entropy pool.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_exporter_build_info	gauge	A metric with a constant '1' value labeled by version, revision, branch, and goversion from which node_exporter was built.
node_filefd_allocated	gauge	File descriptor statistics: allocated.
node_filefd_maximum	gauge	File descriptor statistics: maximum.
node_filesystem_avail_bytes	gauge	Filesystem space available to non-root users in bytes.
node_filesystem_device_error	gauge	Whether an error occurred while getting statistics for the given device.
node_filesystem_files	gauge	Filesystem total file nodes.
node_filesystem_files_free	gauge	Filesystem total free file nodes.
node_filesystem_free_bytes	gauge	Filesystem free space in bytes.
node_filesystem_readonly	gauge	Filesystem read-only status.
node_filesystem_size_bytes	gauge	Filesystem size in bytes.
node_forks_total	counter	Total number of forks.
node_intr_total	counter	Total number of interrupts serviced.
node_ipvs_connections_total	counter	The total number of connections made.
node_ipvs_incoming_bytes_total	counter	The total amount of incoming data.
node_ipvs_incoming_packets_total	counter	The total number of incoming packets.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_ipvs_outgoing_bytes_total	counter	The total amount of outgoing data.
node_ipvs_outgoing_packets_total	counter	The total number of outgoing packets.
node_load1	gauge	1m load average.
node_load15	gauge	15m load average.
node_load5	gauge	5m load average.
node_memory_Active_anon_bytes	gauge	Memory information field Active_anon_bytes.
node_memory_Active_bytes	gauge	Memory information field Active_bytes.
node_memory_Active_file_bytes	gauge	Memory information field Active_file_bytes.
node_memory_AnonHugePages_bytes	gauge	Memory information field AnonHugePages_bytes.
node_memory_AnonPages_bytes	gauge	Memory information field AnonPages_bytes.
node_memory_Bounce_bytes	gauge	Memory information field Bounce_bytes.
node_memory_Buffers_bytes	gauge	Memory information field Buffers_bytes.
node_memory_Cached_bytes	gauge	Memory information field Cached_bytes.
node_memory_CmaFree_bytes	gauge	Memory information field CmaFree_bytes.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_memory_CmaTotal_bytes	gauge	Memory information field CmaTotal_bytes.
node_memory_CommitLimit_bytes	gauge	Memory information field CommitLimit_bytes.
node_memory_Committed_AS_bytes	gauge	Memory information field Committed_AS_bytes.
node_memory_DirectMap2M_bytes	gauge	Memory information field DirectMap2M_bytes.
node_memory_DirectMap4k_bytes	gauge	Memory information field DirectMap4k_bytes.
node_memory_Dirty_bytes	gauge	Memory information field Dirty_bytes.
node_memory_HardwareCorrupted_bytes	gauge	Memory information field HardwareCorrupted_bytes.
node_memory_HugePages_Free	gauge	Memory information field HugePages_Free.
node_memory_HugePages_Rsvd	gauge	Memory information field HugePages_Rsvd.
node_memory_HugePages_Surp	gauge	Memory information field HugePages_Surp.
node_memory_HugePages_Total	gauge	Memory information field HugePages_Total.
node_memory_Hugepagesize_bytes	gauge	Memory information field Hugepagesize_bytes.
node_memory_Inactive_anon_bytes	gauge	Memory information field Inactive_anon_bytes.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_memory_Inactive_bytes	gauge	Memory information field Inactive_bytes.
node_memory_Inactive_file_bytes	gauge	Memory information field Inactive_file_bytes.
node_memory_KernelStack_bytes	gauge	Memory information field KernelStack_bytes.
node_memory_Mapped_bytes	gauge	Memory information field Mapped_bytes.
node_memory_MemAvailable_bytes	gauge	Memory information field MemAvailable_bytes.
node_memory_MemFree_bytes	gauge	Memory information field MemFree_bytes.
node_memory_MemTotal_bytes	gauge	Memory information field MemTotal_bytes.
node_memory_Mlocked_bytes	gauge	Memory information field Mlocked_bytes.
node_memory_NFS_Unstable_bytes	gauge	Memory information field NFS_Unstable_bytes.
node_memory_PageTables_bytes	gauge	Memory information field PageTables_bytes.
node_memory_SReclaimable_bytes	gauge	Memory information field SReclaimable_bytes.
node_memory_SUnreclaim_bytes	gauge	Memory information field SUnreclaim_bytes.
node_memory_ShmemHugePages_bytes	gauge	Memory information field ShmemHugePages_bytes.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_memory_ShmemPmdMapped_bytes	gauge	Memory information field ShmemPmdMapped_bytes.
node_memory_Shmem_bytes	gauge	Memory information field Shmem_bytes.
node_memory_Slab_bytes	gauge	Memory information field Slab_bytes.
node_memory_SwapCached_bytes	gauge	Memory information field SwapCached_bytes.
node_memory_SwapFree_bytes	gauge	Memory information field SwapFree_bytes.
node_memory_SwapTotal_bytes	gauge	Memory information field SwapTotal_bytes.
node_memory_Unevictable_bytes	gauge	Memory information field Unevictable_bytes.
node_memory_VmallocChunk_bytes	gauge	Memory information field VmallocChunk_bytes.
node_memory_VmallocTotal_bytes	gauge	Memory information field VmallocTotal_bytes.
node_memory_VmallocUsed_bytes	gauge	Memory information field VmallocUsed_bytes.
node_memory_WritebackTmp_bytes	gauge	Memory information field WritebackTmp_bytes.
node_memory_Writeback_bytes	gauge	Memory information field Writeback_bytes.
node_netstat_Icmp6_InErrors	counter	Statistic Icmp6InErrors.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_netstat_Icmp6_InMsgs	counter	Statistic Icmp6InMsgs.
node_netstat_Icmp6_OutMsgs	counter	Statistic Icmp6OutMsgs.
node_netstat_Icmp_InErrors	counter	Statistic IcmpInErrors.
node_netstat_Icmp_InMsgs	counter	Statistic IcmpInMsgs.
node_netstat_Icmp_OutMsgs	counter	Statistic IcmpOutMsgs.
node_netstat_Ip6_InOctets	counter	Statistic Ip6InOctets.
node_netstat_Ip6_OutOctets	counter	Statistic Ip6OutOctets.
node_netstat_IpExt_InOctets	counter	Statistic IpExtInOctets.
node_netstat_IpExt_OutOctets	counter	Statistic IpExtOutOctets.
node_netstat_Ip_Forwarding	counter	Statistic IpForwarding.
node_netstat_TcpExt_ListenDrops	counter	Statistic TcpExtListenDrops.
node_netstat_TcpExt_ListenOverflows	counter	Statistic TcpExtListenOverflows.
node_netstat_TcpExt_SyncookiesFailed	counter	Statistic TcpExtSyncookiesFailed.
node_netstat_TcpExt_SyncookiesRecv	counter	Statistic TcpExtSyncookiesRecv.
node_netstat_TcpExt_SyncookiesSent	counter	Statistic TcpExtSyncookiesSent.
node_netstat_TcpExt_TCPSynRetrans	counter	Statistic TcpExtTCPSynRetrans.
node_netstat_TcpExt_TCPTimeouts	counter	Statistic TcpExtTCPTimeouts.
node_netstat_Tcp_ActiveOpens	counter	Statistic TcpActiveOpens.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_netstat_Tcp_CurrEstab	counter	Statistic TcpCurrEstab.
node_netstat_Tcp_InErrs	counter	Statistic TcpInErrs.
node_netstat_Tcp_InSegs	counter	Statistic TcpInSegs.
node_netstat_Tcp_OutRsts	counter	Statistic TcpOutRsts.
node_netstat_Tcp_OutSegs	counter	Statistic TcpOutSegs.
node_netstat_Tcp_PassiveOpens	counter	Statistic TcpPassiveOpens.
node_netstat_Tcp_RetransSegs	counter	Statistic TcpRetransSegs.
node_netstat_Udp6_InDatagrams	counter	Statistic Udp6InDatagrams.
node_netstat_Udp6_InErrors	counter	Statistic Udp6InErrors.
node_netstat_Udp6_NoPorts	counter	Statistic Udp6NoPorts.
node_netstat_Udp6_OutDatagrams	counter	Statistic Udp6OutDatagrams.
node_netstat_Udp6_RcvbufErrors	counter	Statistic Udp6RcvbufErrors.
node_netstat_Udp6_SndbufErrors	counter	Statistic Udp6SndbufErrors.
node_netstat_UdpLite6_InErrors	counter	Statistic UdpLite6InErrors.
node_netstat_UdpLite_InErrors	counter	Statistic UdpLiteInErrors.
node_netstat_Udp_InDatagrams	counter	Statistic UdpInDatagrams.
node_netstat_Udp_InErrors	counter	Statistic UdpInErrors.
node_netstat_Udp_NoPorts	counter	Statistic UdpNoPorts.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_netstat_Udp_OutDatagrams	counter	Statistic UdpOutDatagrams.
node_netstat_Udp_RcvbufErrors	counter	Statistic UdpRcvbufErrors.
node_netstat_Udp_SndbufErrors	counter	Statistic UdpSndbufErrors.
node_network_address_assign_type	gauge	address_assign_type value of **/sys/class/net/**.
node_network_carrier	gauge	carrier value of **/sys/class/net/**.
node_network_carrier_changes_total	counter	carrier_changes_total value of **/sys/class/net/**.
node_network_carrier_down_changes_total	counter	carrier_down_changes_total value of **/sys/class/net/**.
node_network_carrier_up_changes_total	counter	carrier_up_changes_total value of **/sys/class/net/**.
node_network_device_id	gauge	device_id value of **/sys/class/net/**.
node_network_dormant	gauge	dormant value of **/sys/class/net/**.
node_network_flags	gauge	flags value of **/sys/class/net/**.
node_network_iface_id	gauge	iface_id value of **/sys/class/net/**.
node_network_iface_link	gauge	iface_link value of **/sys/class/net/**.
node_network_iface_link_mode	gauge	iface_link_mode value of **/sys/class/net/**.
node_network_info	gauge	Non-numeric data from **/sys/class/net/**, value is always 1.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_network_mtu_bytes	gauge	mtu_bytes value of **/sys/class/net/**.
node_network_name_assign_type	gauge	name_assign_type value of **/sys/class/net/**.
node_network_net_dev_group	gauge	net_dev_group value of **/sys/class/net/**.
node_network_protocol_type	gauge	protocol_type value of **/sys/class/net/**.
node_network_receive_bytes_total	counter	Network device statistic receive_bytes.
node_network_receive_compressed_total	counter	Network device statistic receive_compressed.
node_network_receive_drop_total	counter	Network device statistic receive_drop.
node_network_receive_errs_total	counter	Network device statistic receive_errs.
node_network_receive_fifo_total	counter	Network device statistic receive_fifo.
node_network_receive_frame_total	counter	Network device statistic receive_frame.
node_network_receive_multicast_total	counter	Network device statistic receive_multicast.
node_network_receive_packets_total	counter	Network device statistic receive_packets.
node_network_speed_bytes	gauge	speed_bytes value of **/sys/class/net/**.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_network_transmit_bytes_total	counter	Network device statistic transmit_bytes.
node_network_transmit_carrier_total	counter	Network device statistic transmit_carrier.
node_network_transmit_colls_total	counter	Network device statistic transmit_colls.
node_network_transmit_compressed_total	counter	Network device statistic transmit_compressed.
node_network_transmit_drop_total	counter	Network device statistic transmit_drop.
node_network_transmit_errs_total	counter	Network device statistic transmit_errs.
node_network_transmit_fifo_total	counter	Network device statistic transmit_fifo.
node_network_transmit_packets_total	counter	Network device statistic transmit_packets.
node_network_transmit_queue_length	gauge	transmit_queue_length value of **/sys/class/net/**.
node_network_up	gauge	Value is 1 if operstate is 'up', 0 otherwise.
node_nf_conntrack_entries	gauge	Number of currently allocated flow entries for connection tracking.
node_nf_conntrack_entries_limit	gauge	Maximum size of connection tracking table.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_os_info	gauge	A metric with a constant '1' value labeled by build_id, id, id_like, image_id, image_version, name, pretty_name, variant, variant_id, version, version_codename, version_id.
node_os_version	gauge	Metric containing the major.minor part of the OS version.
node_power_supply_info	gauge	info of **/sys/class/power_supply/<power_supply>**.
node_power_supply_online	gauge	online value of **/sys/class/power_supply/<power_supply>**.
node_procs_blocked	gauge	Number of processes blocked waiting for I/O to complete.
node_procs_running	gauge	Number of processes in runnable state.
node_schedstat_running_seconds_total	counter	Number of seconds CPU spent running a process.
node_schedstat_timeslices_total	counter	Number of timeslices executed by CPU.
node_schedstat_waiting_seconds_total	counter	Number of seconds spent by processing waiting for this CPU.
node_scrape_collector_duration_seconds	gauge	node_exporter: Duration of a collector scrape.
node_scrape_collector_success	gauge	node_exporter: Whether a collector succeeded.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_sockstat_FRAG6_inuse	gauge	Number of FRAG6 sockets in state inuse.
node_sockstat_FRAG6_memory	gauge	Number of FRAG6 sockets in state memory.
node_sockstat_FRAG_inuse	gauge	Number of FRAG sockets in state inuse.
node_sockstat_FRAG_memory	gauge	Number of FRAG sockets in state memory.
node_sockstat_RAW6_inuse	gauge	Number of RAW6 sockets in state inuse.
node_sockstat_RAW_inuse	gauge	Number of RAW sockets in state inuse.
node_sockstat_TCP6_inuse	gauge	Number of TCP6 sockets in state inuse.
node_sockstat_TCP_alloc	gauge	Number of TCP sockets in state alloc.
node_sockstat_TCP_inuse	gauge	Number of TCP sockets in state inuse.
node_sockstat_TCP_mem	gauge	Number of TCP sockets in state mem.
node_sockstat_TCP_mem_bytes	gauge	Number of TCP sockets in state mem_bytes.
node_sockstat_TCP_orphan	gauge	Number of TCP sockets in state orphan.
node_sockstat_TCP_tw	gauge	Number of TCP sockets in state tw.

Table 13: Cluster Node Metric List ***(Continued)***

Metric Name	Type	Description
node_sockstat_UDP6_inuse	gauge	Number of UDP6 sockets in state inuse.
node_sockstat_UDPLITE6_inuse	gauge	Number of UDPLITE6 sockets in state inuse.
node_sockstat_UDPLITE_inuse	gauge	Number of UDPLITE sockets in state inuse.
node_sockstat_UDP_inuse	gauge	Number of UDP sockets in state inuse.
node_sockstat_UDP_mem	gauge	Number of UDP sockets in state mem.
node_sockstat_UDP_mem_bytes	gauge	Number of UDP sockets in state mem_bytes.
node_sockstat_sockets_used	gauge	Number of IPv4 sockets in use.
node_softnet_dropped_total	counter	Number of dropped packets.
node_softnet_processed_total	counter	Number of processed packets
node_softnet_times_squeezed_total	counter	Number of times processing packets ran out of quota.
node_textfile_scrape_error	gauge	1 if there was an error opening or reading a file, 0 otherwise.
node_time_clocksource_available_info	gauge	Available clocksources read from **/sys/devices/system/clocksource**.
node_time_clocksource_current_info	gauge	Current clocksource read from **/sys/devices/system/clocksource**.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_time_seconds	gauge	System time in seconds since epoch (1970).
node_time_zone_offset_seconds	gauge	System time zone offset in seconds.
node_timex_estimated_error_seconds	gauge	Estimated error in seconds.
node_timex_frequency_adjustment_ratio	gauge	Local clock frequency adjustment.
node_timex_loop_time_constant	gauge	Phase-locked loop time constant.
node_timex_maxerror_seconds	gauge	Maximum error in seconds.
node_timex_offset_seconds	gauge	Time offset in between local system and reference clock.
node_timex_pps_calibration_total	counter	Pulse per second count of calibration intervals.
node_timex_pps_error_total	counter	Pulse per second count of calibration errors.
node_timex_pps_frequency_hertz	gauge	Pulse per second frequency.
node_timex_pps_jitter_seconds	gauge	Pulse per second jitter.
node_timex_pps_jitter_total	counter	Pulse per second count of jitter limit exceeded events.
node_timex_pps_shift_seconds	gauge	Pulse per second interval duration.
node_timex_pps_stability_exceeded_total	counter	Pulse per second count of stability limit exceeded events.
node_timex_pps_stability_hertz	gauge	Pulse per second stability, average of recent frequency changes.

Table 13: Cluster Node Metric List *(Continued)*

Metric Name	Type	Description
node_timex_status	gauge	Value of the status array bits.
node_timex_sync_status	gauge	Is clock synchronized to a reliable server (1 = yes, 0 = no).
node_timex_tai_offset_seconds	gauge	International Atomic Time (TAI) offset.
node_timex_tick_seconds	gauge	Seconds between clock ticks.
node_udp_queues	gauge	Number of allocated memory in the kernel for UDP datagrams in bytes.
node_uname_info	gauge	Labeled system information as provided by the uname system call.
node_vmstat_oom_kill	counter	**/proc/vmstat** information field oom_kill.
node_vmstat_pgfault	counter	**/proc/vmstat** information field pgfault.
node_vmstat_pgmajfault	counter	**/proc/vmstat** information field pgmajfault.
node_vmstat_pgpgin	counter	**/proc/vmstat** information field pgpgin.
node_vmstat_pgpgout	counter	**/proc/vmstat** information field pgpgout.
node_vmstat_pswpin	counter	**/proc/vmstat** information field pswpin.
node_vmstat_pswpout	counter	**/proc/vmstat** information field pswpout.

RELATED DOCUMENTATION

Contrail Networking Analytics | 93
Contrail Networking Metric List | 99
Kubernetes Metric List | 113
Contrail Networking Alert List | 169

Contrail Networking Alert List

Table 14: Contrail Networking Alert List

Alert Name	Severity	Description
VRouterConnectionDown	major	VRouter <name> <connection_type> connection to <connection_id> is down.
VRouterNonFunctional	major	VRouter <name> is non-functional.
ControllerNonFunctional	major	Controller <name> is non-functional.
ControllerConnectionDown	major	Controller <name> <connection_type> connection to <connection_id> is down.
ControllerDBConnectionDown	major	Controller <name> connection to database is down.
AlertmanagerFailedReload	critical	Reloading an Alertmanager configuration has failed.
AlertmanagerMembersInconsistent	critical	A member of an Alertmanager cluster has not found all other cluster members.
AlertmanagerFailedToSendAlerts	warning	An Alertmanager instance failed to send notifications.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
AlertmanagerClusterFailedToSendAlerts	critical	All Alertmanager instances in a cluster failed to send notifications to a critical integration.
AlertmanagerClusterFailedToSendAlerts	warning	All Alertmanager instances in a cluster failed to send notifications to a non-critical integration.
AlertmanagerConfigInconsistent	critical	Alertmanager instances within the same cluster have different configurations.
AlertmanagerClusterDown	critical	Half or more of the Alertmanager instances within the same cluster are down.
AlertmanagerClusterCrashlooping	critical	Half or more of the Alertmanager instances within the same cluster are crashlooping.
ConfigReloaderSidecarErrors	warning	config-reloader sidecar has not had a successful reload for 10m.
etcdInsufficientMembers	critical	etcd cluster "<name>": insufficient members (<value>).
etcdNoLeader	critical	etcd cluster "<name>": member <instance> has no leader.
etcdHighNumberOfLeaderChanges	warning	etcd cluster "<name>": instance <instance> has seen <value> leader changes within the last hour.
etcdHighNumberOfFailedGRPCRequests	warning	etcd cluster "<name>": <value>% of requests for <grpc_method> failed on etcd instance <instance>.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
etcdHighNumberOfFailedGRPCRequests	critical	etcd cluster "<name>": <value>% of requests for <grpc_method> failed on etcd instance <instance>.
etcdGRPCRequestsSlow	critical	etcd cluster "<name>": gRPC requests to <grpc_method> are taking <value>s on etcd instance <instance>.
etcdMemberCommunicationSlow	warning	etcd cluster "<name>": member communication with <name> is taking <value>s on etcd instance <instance>.
etcdHighNumberOfFailedProposals	warning	etcd cluster "<name>": <value> proposal failures within the last hour on etcd instance <instance>.
etcdHighFsyncDurations	warning	etcd cluster "<name>": 99th percentile fsync durations are <value>s on etcd instance <instance>.
etcdHighCommitDurations	warning	etcd cluster "<name>": 99th percentile commit durations <value>s on etcd instance <instance>.
etcdHighNumberOfFailedHTTPRequests	warning	<value>% of requests for <method> failed on etcd instance <instance>.
etcdHighNumberOfFailedHTTPRequests	critical	<value>% of requests for <method> failed on etcd instance <instance>.
etcdHTTPRequestsSlow	warning	etcd instance <instance> HTTP requests to <method> are slow.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
`TargetDown`	warning	One or more targets are unreachable.
`KubeAPIErrorBudgetBurn`	critical	The API server is burning too much error budget.
`KubeAPIErrorBudgetBurn`	warning	The API server is burning too much error budget.
`KubeStateMetricsListErrors`	critical	`kube-state-metrics` is experiencing errors in list operations.
`KubeStateMetricsWatchErrors`	critical	`kube-state-metrics` is experiencing errors in watch operations.
`KubeStateMetricsShardingMismatch`	critical	`kube-state-metrics` sharding is misconfigured.
`KubeStateMetricsShardsMissing`	critical	`kube-state-metrics` shards are missing.
`KubePodCrashLooping`	warning	Pod is crash looping.
`KubePodNotReady`	warning	Pod has been in a non-ready state for more than 15 minutes.
`KubeDeploymentGenerationMismatch`	warning	Deployment generation mismatch due to possible roll-back.
`KubeDeploymentReplicasMismatch`	warning	Deployment has not matched the expected number of replicas.
`KubeStatefulSetReplicasMismatch`	warning	Deployment has not matched the expected number of replicas.
`KubeStatefulSetGenerationMismatch`	warning	`StatefulSet` generation mismatch due to possible roll-back.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
`KubeStatefulSetUpdateNotRolledOut`	warning	`StatefulSet` update has not been rolled out.
`KubeDaemonSetRolloutStuck`	warning	`DaemonSet` rollout is stuck.
`KubeContainerWaiting`	warning	Pod container waiting longer than 1 hour.
`KubeDaemonSetNotScheduled`	warning	`DaemonSet` pods are not scheduled.
`KubeDaemonSetMisScheduled`	warning	`DaemonSet` pods are misscheduled.
`KubeJobCompletion`	warning	Job did not complete in time.
`KubeJobFailed`	warning	Job failed to complete.
`KubeHpaReplicasMismatch`	warning	HPA has not matched desired number of replicas.
`KubeHpaMaxedOut`	warning	HPA is running at max replicas.
`KubeCPUOvercommit`	warning	Cluster has overcommitted CPU resource requests.
`KubeMemoryOvercommit`	warning	Cluster has overcommitted CPU resource requests.
`KubeCPUQuotaOvercommit`	warning	Cluster has overcommitted CPU resource requests.
`KubeMemoryQuotaOvercommit`	warning	Cluster has overcommitted memory resource requests.
`KubeQuotaAlmostFull`	info	Namespace quota is going to be full.
`KubeQuotaFullyUsed`	info	Namespace quota is fully used.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
`KubeQuotaExceeded`	warning	Namespace quota has exceeded the limits.
`CPUThrottlingHigh`	info	Processes experience elevated CPU throttling.
`KubePersistentVolumeFillingUp`	critical	`PersistentVolume` is filling up.
`KubePersistentVolumeFillingUp`	warning	`PersistentVolume` is filling up.
`KubePersistentVolumeErrors`	critical	`PersistentVolume` is having issues with provisioning.
`KubeVersionMismatch`	warning	Different semantic versions of Kubernetes components running.
`KubeClientErrors`	warning	Kubernetes API server client is experiencing errors.
`KubeClientCertificateExpiration`	warning	Client certificate is about to expire.
`KubeClientCertificateExpiration`	critical	Client certificate is about to expire.
`KubeAggregatedAPIErrors`	warning	Kubernetes aggregated API has reported errors.
`KubeAggregatedAPIDown`	warning	Kubernetes aggregated API is down.
`KubeAPIDown`	critical	Target disappeared from Prometheus target discovery.
`KubeAPITerminatedRequests`	warning	The Kubernetes `apiserver` has terminated `<value>` of its incoming requests.
`KubeControllerManagerDown`	critical	Target disappeared from Prometheus target discovery.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
`KubeProxyDown`	critical	Target disappeared from Prometheus target discovery.
`KubeNodeNotReady`	warning	Node is not ready.
`KubeNodeUnreachable`	warning	Node is unreachable.
`KubeletTooManyPods`	info	Kubelet is running at capacity.
`KubeNodeReadinessFlapping`	warning	Node readiness status is flapping.
`KubeletPlegDurationHigh`	warning	Kubelet Pod Lifecycle Event Generator is taking too long to relist.
`KubeletPodStartUpLatencyHigh`	warning	Kubelet Pod startup latency is too high.
`KubeletClientCertificateExpiration`	warning	Kubelet client certificate is about to expire.
`KubeletClientCertificateExpiration`	critical	Kubelet client certificate is about to expire.
`KubeletServerCertificateExpiration`	warning	Kubelet server certificate is about to expire.
`KubeletServerCertificateExpiration`	critical	Kubelet server certificate is about to expire.
`KubeletClientCertificateRenewalErrors`	warning	Kubelet has failed to renew its client certificate.
`KubeletServerCertificateRenewalErrors`	warning	Kubelet has failed to renew its server certificate.
`KubeletDown`	critical	Target disappeared from Prometheus target discovery.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
`KubeSchedulerDown`	critical	Target disappeared from Prometheus target discovery.
`NodeFilesystemSpaceFillingUp`	warning	Filesystem is predicted to run out of space within the next 24 hours.
`NodeFilesystemSpaceFillingUp`	critical	Filesystem is predicted to run out of space within the next 4 hours.
`NodeFilesystemAlmostOutOfSpace`	warning	Filesystem has less than 5% space left.
`NodeFilesystemAlmostOutOfSpace`	critical	Filesystem has less than 3% space left.
`NodeFilesystemFilesFillingUp`	warning	Filesystem is predicted to run out of inodes within the next 24 hours.
`NodeFilesystemFilesFillingUp`	critical	Filesystem is predicted to run out of inodes within the next 4 hours.
`NodeFilesystemAlmostOutOfFiles`	warning	Filesystem has less than 5% inodes left.
`NodeFilesystemAlmostOutOfFiles`	critical	Filesystem has less than 3% inodes left.
`NodeNetworkReceiveErrs`	warning	Network interface is reporting many receive errors.
`NodeNetworkTransmitErrs`	warning	Network interface is reporting many transmit errors.
`NodeHighNumberConntrackEntriesUsed`	warning	Number of `conntrack` are getting close to the limit.
`NodeTextFileCollectorScrapeError`	warning	Node Exporter text file collector failed to scrape.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
NodeClockSkewDetected	warning	Clock skew detected.
NodeClockNotSynchronising	warning	Clock not synchronising.
NodeRAIDDegraded	critical	RAID Array is degraded.
NodeRAIDDiskFailure	warning	Failed device in RAID array.
NodeFileDescriptorLimit	warning	Kernel is predicted to exhaust file descriptors limit soon.
NodeFileDescriptorLimit	critical	Kernel is predicted to exhaust file descriptors limit soon.
NodeNetworkInterfaceFlapping	warning	Network interface is often changing its status.
PrometheusBadConfig	critical	Failed Prometheus configuration reload.
PrometheusNotificationQueueRunningFull	warning	Prometheus alert notification queue predicted to run full in less than 30m.
PrometheusErrorSendingAlertsToSomeAlertmanagers	warning	Prometheus has encountered more than 1% errors sending alerts to a specific Alertmanager.
PrometheusNotConnectedToAlertmanagers	warning	Prometheus is not connected to any Alertmanagers.
PrometheusTSDBReloadsFailing	warning	Prometheus has issues reloading blocks from disk.
PrometheusTSDBCompactionsFailing	warning	Prometheus has issues compacting blocks.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
`PrometheusNotIngestingSamples`	warning	Prometheus is not ingesting samples.
`PrometheusDuplicateTimestamps`	warning	Prometheus is dropping samples with duplicate timestamps.
`PrometheusOutOfOrderTimestamps`	warning	Prometheus drops samples with out-of-order timestamps.
`PrometheusRemoteStorageFailures`	critical	Prometheus fails to send samples to remote storage.
`PrometheusRemoteWriteBehind`	critical	Prometheus remote write is behind.
`PrometheusRemoteWriteDesiredShards`	warning	Prometheus remote write desired shards calculation wants to run more than configured max shards.
`PrometheusRuleFailures`	critical	Prometheus is failing rule evaluations.
`PrometheusMissingRuleEvaluations`	warning	Prometheus is missing rule evaluations due to slow rule group evaluation.
`PrometheusTargetLimitHit`	warning	Prometheus has dropped targets because some scrape configs have exceeded the targets limit.
`PrometheusLabelLimitHit`	warning	Prometheus has dropped targets because some scrape configs have exceeded the labels limit.
`PrometheusTargetSyncFailure`	critical	Prometheus has failed to sync targets.
`PrometheusErrorSendingAlertsToAnyAlertmanager`	critical	Prometheus encounters more than 3% errors sending alerts to any Alertmanager.

Table 14: Contrail Networking Alert List *(Continued)*

Alert Name	Severity	Description
`PrometheusOperatorListErrors`	warning	Errors while performing list operations in controller.
`PrometheusOperatorWatchErrors`	warning	Errors while performing list operations in controller.
`PrometheusOperatorSyncFailed`	warning	Last controller reconciliation failed.
`PrometheusOperatorReconcileErrors`	warning	Errors while reconciling controller.
`PrometheusOperatorNodeLookupErrors`	warning	Errors while reconciling Prometheus.
`PrometheusOperatorNotReady`	warning	Prometheus operator not ready.
`PrometheusOperatorRejectedResources`	warning	Resources rejected by Prometheus operator.

RELATED DOCUMENTATION

Contrail Networking Analytics | 93
Contrail Networking Metric List | 99
Kubernetes Metric List | 113
Cluster Node Metric List | 152

vRouter Session Analytics in Contrail Networking

IN THIS SECTION

Juniper Networks supports the collection, storage, and query for vRouter traffic in environments using Cloud-Native Contrail® Networking™ Release 22.1 or later in a Kubernetes-orchestrated environment.

Collector Module

Contrail Networking collects user visible entities (UVEs) and traffic information (session) for traffic analysis and troubleshooting. The `collector` module stores these objects and provides APIs to access the collected information.

The Contrail Networking vRouter agent exports data records to the collector when events are created or deleted.

Collector Deployment

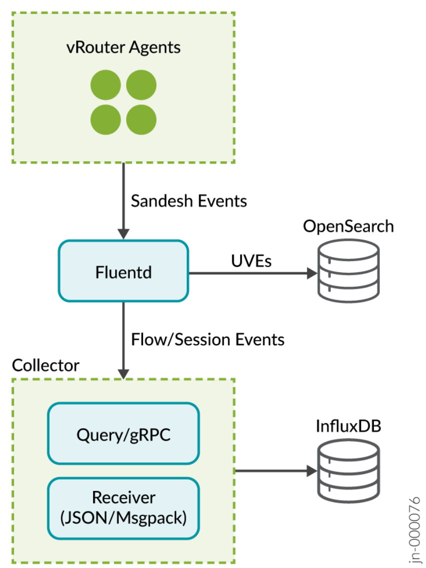
The following components are installed in the Contrail cluster in the `contrail` namespace (NS):

- **Collector Microservice**—Collects incoming events.
- **InfluxDB**—A time series database built specifically for storing time series data. Works with Grafana as a visualization tool for time series data.
- **Fluentd**—Logging agent that performs log collection, parsing, and distribution to other services such asOpenSearch.
- **OpenSearch**—OpenSearch is the search and analytics engine in the AWS OpenSearch Stack, providing real-time search and analytics for all types of data.
- **OpenSearch Dashboards** —User interface that lets you visualize your OpenSearch data and navigate the OpenSearch Stack.

Data Collection

Figure 8 on page 181 shows the data collection.

Figure 8: Cloud-Native Contrail Collector: Event and Log Ingestion



UVEs

UVEs are stored in OpenSearch in an index named by the name of the UVE.

Session

Session records are stored in InfluxDB. These records are pushed as events from all agents. This data is downsampled for longer duration. Retention periods of live, downsampled table, and downsampling windows are configurable using the configuration.

Table 15: Session Records Information

Column	Filterable	Detail
`vn`	Yes	Client Virtual Network
`vmi`	Yes	Interface

Table 15: Session Records Information *(Continued)*

Column	Filterable	Detail
remote_vn	Yes	Server Virtual Network
vrouter_ip	Yes	Agent IP
local_ip	Yes	Client IP
client_port	Yes	Client Port
remote_ip	Yes	Server IP
server_port	Yes	Server Port
protocol	Yes	Protocol
label.local.<label-name>	Yes	Client Pod Labels (For example, client pod with label site maps to label.local.site tag in database.)
label.remote.<label-name>	Yes	Server Pod Labels
forward_sampled_bytes	No	Bytes Sent
forward_sampled_pkts	No	Packets Sent
reverse_sampled_bytes	No	Bytes Received
reverse_sampled_pkts	No	Packets Received
total_bytes	No	Total Bytes Exchanged

Configure Data Collection

To configure vRouter agents to send `SessionEndpoint` messages to the `fluentd` service, run the following three commands. Replace `<cluster-ip>` with the cluster IP address of the `fluentd` service in the contrail-analytics namespace.

```
kubectl -n contrail patch vrouter contrail-vrouter-masters --type=merge -p '{"spec":{"agent":
{"default":{"collectors":["<cluster-ip>:24224"]}}}}'
```

```
kubectl -n contrail patch vrouter contrail-vrouter-nodes --type=merge -p '{"spec":{"agent":
{"default":{"collectors":["<cluster-ip>:24224"]}}}}'
```

```
kubectl -n contrail patch gvc default-global-vrouter-config --type=merge -p '{"spec":
{"flowExportRate": 10000}}'
```

After running the three configuration commands, restart vRouter for the configuration to take effect. To restart vRouter, run the following command:

```
kubectl -n contrail delete $(kubectl get pods -l 'app in (contrail-vrouter-masters, contrail-
vrouter-nodes)' -n contrail -o name)
```

Collector Query

The `collector` modules provide a query interface for access.

Run a Query

Example Query

The following query gets total bytes exchanged between unique source-destination pairs (by labels) in the `contrail-analytics` namespace:

```
{
  "granularity": 3600,
  "column": [
    {
      "name": "total_bytes",
      "aggregation": "sum"
    },
    {
      "name": "/^label.*/",
      "regex": true
    }
  ],
  "skip_columns": [
    "label.remote.pod-template-hash",
    "label.local.pod-template-hash"
  ],
  "range":{
    "start_time": -3600
  },
  "filter": [
    {
      "field": "label.local.namespace",
      "operator": "==",
      "value": "contrail-analytics"
    },
    {
      "field": "label.remote.namespace",
      "operator": "==",
      "value": "contrail-analytics"
    }
  ]
}
```

Example Query Response

```
{
  "status": "success",
  "total": 5,
```

```
  "data": {
    "resultType": "matrix",
    "result": [
      {
        "metric": {
          "label.local.namespace": "contrail-analytics",
          "label.remote.app": "collector",
          "label.remote.namespace": "contrail-analytics"
        },
        "fields": [
          "_time",
          "total_bytes"
        ],
        "values": [
          [
            1645768800,
            31012095
          ]
        ]
      },
      {
        "metric": {
          "label.local.namespace": "contrail-analytics",
          "label.remote.app": "opensearch",
          "label.remote.chart": "opensearch",
          "label.remote.controller-revision-hash": "opensearch-7fcc8df678",
          "label.remote.namespace": "contrail-analytics",
          "label.remote.release": "contrail-analytics"
        },
        "fields": [
          "_time",
          "total_bytes"
        ],
        "values": [
          [
            1645768800,
            221493
          ]
        ]
      },
      {
        "metric": {
          "label.local.controller-revision-hash": "5599999fc7",
```

```
        "label.local.namespace": "contrail-analytics",
        "label.local.pod-template-generation": "1",
        "label.remote.namespace": "contrail-analytics"
      },
      "fields": [
        "_time",
        "total_bytes"
      ],
      "values": [
        [
          1645768800,
          23349247
        ]
      ]
    },
    {
      "metric": {
        "label.local.app": "collector",
        "label.local.namespace": "contrail-analytics",
        "label.remote.controller-revision-hash": "influxdb-7bdd86f8c",
        "label.remote.namespace": "contrail-analytics"
      },
      "fields": [
        "_time",
        "total_bytes"
      ],
      "values": [
        [
          1645768800,
          10412552
        ]
      ]
    },
    {
      "metric": {
        "label.local.app": "opensearch-dashboards",
        "label.local.namespace": "contrail-analytics",
        "label.local.release": "contrail-analytics",
        "label.remote.app": "opensearch",
        "label.remote.chart": "opensearch",
        "label.remote.controller-revision-hash": "opensearch-7fcc8df678",
        "label.remote.namespace": "contrail-analytics",
        "label.remote.release": "contrail-analytics"
```

```
          },
          "fields": [
            "_time",
            "total_bytes"
          ],
          "values": [
            [
              1645768800,
              25152
            ]
          ]
        }
      ]
    }
  }
```


RELATED DOCUMENTATION

Contrail Networking Analytics | 93

Centralized Logging | 187

Centralized Logging

IN THIS SECTION

Juniper Networks supports centralized logging using Cloud-Native Contrail® Networking™ Release 22.1 or later in a Kubernetes-orchestrated environment.

Benefits of Centralized Logging

- The centralization of all platform logs eases troubleshooting. Allowing you (the administrator) to take a holistic view of events, or outages, across the many components within the deployment.
- You have one portal, allowing you to monitor, view, filter, and search for events across all platform components.

Overview: Centralized Logging

Instead of browsing through individual log files, collected logs from all components of Contrail Networking are available to the administrator in a centralized location. The centralized location provides the ability to correlate the log files from multiple software components. For security, strict logging exists for all create, read, update, and delete (CRUD) actions. An administrator performs these actions with individual access credentials so that individuals can be identified.

AWS OpenSearch Stack, an open source log collector and analyzer framework, provides out-of-box log collection and analysis functionality. The OpenSearch stack allows a single portal for analyzing logs from Contrail Networking. OpenSearch stack also analyzes logs from other software components and platforms deployed in the cluster. Examples include Linux OS logs, Kubernetes logs, and software components such as virtualized network functions (VNFs) and container network functions (CNFs).

OpenSearch Stack includes:

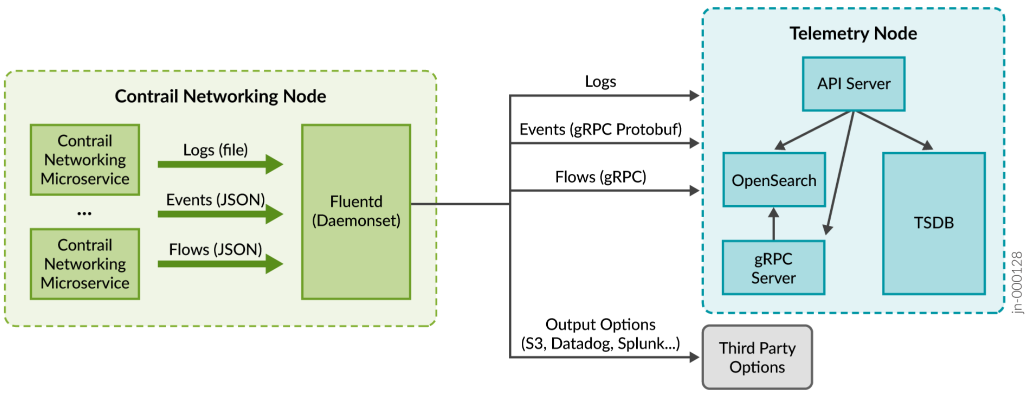
- OpenSearch—Real-time and scalable search engine which allows for full-text and structured search, as well as analytics. This search engine indexes and searches through large volumes of log data.
- OpenSearch Dashboard—Allows you to explore your OpenSearch log data through a web interface, and build dashboards and queries.
- Fluentd—Logging agent that performs log collection, parsing, and distribution to other services such as OpenSearch.
- Fluent Bit—Log processor and forwarder that collects data, such as metrics and logs from different sources. High throughput with low CPU and memory usage. Fluent Bit is installed in every workload cluster.

The logging components are included and deployed in the optional telemetry node deployment. Installation commands are integrated in the telemetry installation.

Logs, Events, and Flows with Fluentd

Fluentd collects logs, events, and flows running on each Contrail Networking node. Fluentd is the logging agent that performs log collection, parsing, and distribution to other services such as OpenSearch.

Figure 9: Logs, Events, and Flows with Fluentd



Logs

Logs are collected from log files or `stdout/stderr` data streams and directed to the OpenSearch library stack with cluster quorum. Each Contrail Networking node (configuration, control, compute, Web UI, and telemetry node) runs Fluent Bit or Fluentd to collect logs. The logs are sent to multiple configured sinks, such as OpenSearch. Fluentd supports multiple output options to send collected logs.

- Control and compute nodes generate unstructured and structured logs through the Sandesh library. The Contrail Networking Sandesh library generates structured JSON files.
- Configuration, Web UI, and telemetry node components produce standard logs to files or to `stdout/stderr`, that are then sent to Fluentd or Fluent Bit.

Multiple Kubernetes clusters in any Contrail Networking cluster or in multiple Contrail Networking clusters are able to connect with a Fluentd/OpenSearch monitoring component.

Events

vRouter agent and control node produce events through Sandesh. The Sandesh library produces JSON structured data and sends those files to the configured options. Configured options are `stdout`, file, or TCP port (Fluentd). Fluentd is configured with multiple output options to send data either to OpenSearch or to the telemetry node's gRPC server. The telemetry node keeps cache for the latest status events.

Flows

The vRouter agent produces flow meta at regular configured intervals. Configuration options for flow data generation supported by the vRouter agent are syslogs, JSON structure, and the default Sandesh.

RELATED DOCUMENTATION

Port-Based Mirroring

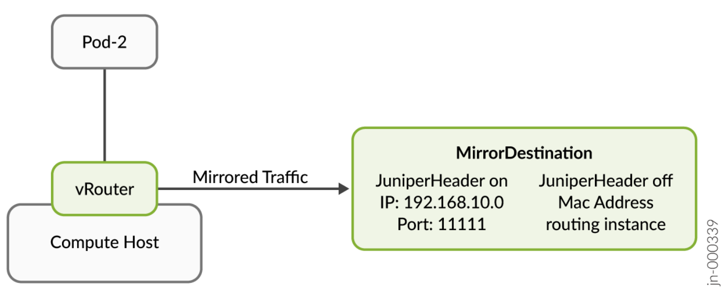
SUMMARY

This section describes port-based mirroring in Juniper Cloud-Native Contrail® Networking™ Release 22.2 and later in a Kubernetes-orchestrated environment.

IN THIS SECTION

Overview: Port-Based Mirroring

Figure 10: Cloud-Native Contrail Networking Port-Based Mirror Topology



Port mirroring sends network traffic from defined ports to a network analyzer where you can monitor and analyze the data. In Cloud-Native Contrail Networking, the following is supported:

- Mirroring configuration is primarily driven from the pod configuration for both the receiver and interface being mirrored. You don't need to configure the virtual machine interface (VMI) directly.
- Mirroring configuration involves creating a `mirrorDestination` resource and associating the `mirrorDestination` resource to the pod interface to be mirrored.
- `MirrorDestination` identifies the mirrored traffic receiver pod and interface. When `juniperHeader` is enabled, receiver pod IP address and port are used. When `juniperHeader` is disabled, receiver pod MAC address `routingInstance` is used to forward mirrored traffic.
- A `mirrorDestination` can be associated with multiple VMIs to be mirrored.
- `MirrorDestination` resource defines the mirrored traffic receiver such as, IP address, port used for receiving mirrored traffic, Juniper header configuration, dynamic or static next-hop, and so on.
- A pod interface to be mirrored can be configured when creating the pod or by editing the pod.

Example: Configure Port-Based Mirroring

The following procedure is an example configuration that creates a `MirrorDestination` resource and specifies the `mirrorDestination` resource name, for example `mirrordestinationprofile1`, on the interface to be mirrored.

1. Use the `MirrorDestination` YAML to create a `mirrorDestination` resource by adding multiple destination pods with the label `core.juniper.net/analyzer-pod-selector: analyzerpod`.

   - `MirrorDestination` resource uses the label `core.juniper.net/analyzer-pod-selector: analyzerpod` to calculate and determine the mirrored traffic pod receiver.

   Example `MirrorDestination` YAML file.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: MirrorDestination
metadata:
  name: mirrordestinationprofile1
  labels:
    core.juniper.net/analyzer-pod-selector: analyzerpod
spec:
  trafficDirection: <ingress|egress|both>
  juniperHeader: <boolean>
```

```
  udpPort: <integer>
  staticNhHeader: <null for dynamic nh|vtep tunnel destip, mac, vxlanid for static nh>
  nextHopMode: <static|dynamic>
  nicAssistedMirroring: <boolean>
    nicAssistedVlanID:
  staticNextHopHeader:
    vTEPDestinationIP:
    vTEPDestinationMac:
    vxlanID:
```

When you deploy the YAML file, there could be multiple pods matching the label `analyzerpod`. First matching pod is selected as the mirrored traffic receiver. The selected pod remains sticky until the pod or interface is no longer available.

Following is the analyzer pod YAML file with label `analyzerpod`, indicating `MirrorDestination` can use this pod.

- Note the label value for `core.juniper.net/analyzer-podanalyzerpod` is the same as specified in the `MirrorDestination` YAML file.
- The `mirrorDestination` controller uses this label to calculate the `analyzer_ip`, `macaddress`, and `routinginstance`.
- The pod interface to be used is specified in annotation below:

  `core.juniper.net/analyzer-interface: true`

  To indicate default pod interface, it is specified directly under annotations. For a custom VN interface, it is specified in `cni-args` of the network. The example `Pod/analyzerpod` YAML file shows both examples.
- `core.juniper.net/analyzer-interface: true` indicates the `vn-1` pod interface will receive mirrored traffic.

Example `Pod/analyzerpod` YAML file.

```
apiVersion: v1
kind: Pod
metadata:
  name: analyzerpod
  namespace: mirror-ns
  labels:
    core.juniper.net/analyzer-pod: analyzerpod
  annotations:
    core.juniper.net/analyzer-interface: "true"
    k8s.v1.cni.cncf.io/networks: |
```

```
        [
          {
            "name": "vn-1",
            "namespace": "mirror-ns",
            "cni-args": {
              "core.juniper.net/analyzer-interface": "true"
            }
          }
        ]
```

2. Add the pod annotations and specify the `mirroringDestination` resource name on the interface to be mirrored.

   In the following example YAML file, we are enabling mirroring on the pod `vn-1` interface. We specify the `mirrorDestination` resource name `mirrordestinationprofile1` on the interface to be mirrored.

   Example `Pod/mirrored-pod` YAML file.

```
apiVersion: v1
kind: Pod
metadata:
  name: mirrored-pod
  namespace: mirror-ns
  annotations:
    core.juniper.net/mirror-destination: "mirrordestinationprofile1"
    k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name": "vn-1",
          "namespace": "mirror-ns",
          "cni-args": {
            "core.juniper.net/mirror-destination": "mirrordestinationprofile1"
          }
        }
      ]
```

Summary

SUMMARY

This section describes configuration changes for port-based mirroring in Cloud-Native Contrail Networking Release 22.2.

IN THIS SECTION

From the analyzer pod annotations and labels, the VM and VMI are associated with the pod to be used in the `mirrorDestination` controller.

Analyzer VM Labels:

The `VirtualMachine` resource corresponding to the pod will have the label `core.juniper.net/analyzer-pod` label.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualMachine
metadata:
  annotations:
    kube-manager.juniper.net/pod-cluster-name: contrail-k8s-kubemanager-ocp-kparmar-6mpccd
    kube-manager.juniper.net/pod-name: analyzerpod
    kube-manager.juniper.net/pod-namespace: multinode-ns
  labels:
    core.juniper.net/analyzer-pod: analyzerpod
```

Analyzer VMI Labels:

The `VirtualMachineInterface` resource for the analyzer pod will have the label `core.juniper.net/analyzer-interface`.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualMachineInterface
metadata:
  annotations:
    index: 0/1
    interface: eth0
    kube-manager.juniper.net/pod-cluster-name: contrail-k8s-kubemanager-ocp-kparmar-6mpccd
    kube-manager.juniper.net/pod-name: analyzerpod
    kube-manager.juniper.net/pod-namespace: multinode-ns
```

```
  labels:
    core.juniper.net/analyzer-interface: ""
```

Source VMI Label indicating `mirrorDestination`:

Source `VirtualMachineInterface` corresponding to the pod interface being mirrored will have label `core.juniper.net/mirror-destination`. And the annotations will have the mirror configuration.

```
apiVersion: core.contrail.juniper.net/v1alpha1
kind: VirtualMachineInterface
metadata:
  annotations:
    core.juniper.net/mirroring-configuration:
'{"analyzer_name":"mirrordestinationprofile1","analyzer_ip_address":"10.128.0.200","analyzer_maca
ddress":"02:76:6c:25:f2:8c","ri":"default-
    domain:contrail-k8s-kubemanager-ocp-kparmar-6mpccd-contrail:default-podnetwork:default-
podnetwork"}'
  labels:
    core.juniper.net/mirror-destination: mirrordestinationprofile1
```

Configurable Categories of Metrics Collection and Reporting (Tech Preview)

SUMMARY

In Cloud-Native Contrail® Networking™ Release 22.2, you can enable and disable selected metrics for exporting.

IN THIS SECTION

Overview: Configurable Categories of Metrics Collection and Reporting

To provide more flexibility in the telemetry export component, Cloud-Native Contrail Networking Release 22.2 introduces a new Kubernetes custom resource: `MetricGroup`. `MetricGroup` allows you to enable or disable selected metrics for exporting.

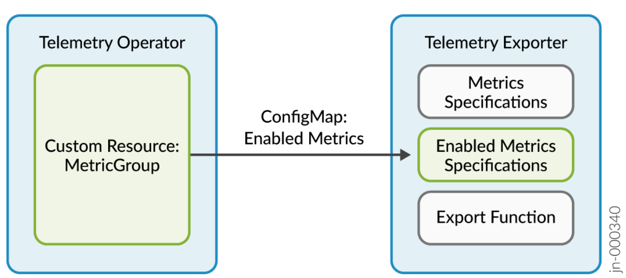
- `MetricGroup` contains and manages a set of metrics for exporting.
- Metrics are grouped by their category. You can choose to enable or disable the metric export function at the group level.
- `MetricGroup` is implemented through a Kubernetes custom resource.

`MetricGroup` provides fine-grained control on what metrics the system collects and reports. You can turn on and off subset of metrics reporting. Sometimes administrators want to collect only a subset of metrics for efficiency and the lightest weight possible deployment.

NOTE: This feature is classified as a Juniper CN2 Technology Preview feature. These features are "as is" and are for 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.

For additional information, see "Juniper CN2 Technology Previews (Tech Previews)" on page 201 or contact Juniper Support.

Figure 11: Metrics Collection and Reporting Architecture



Telemetry Operator, see Figure 11 on page 196, monitors any change of metric groups. Based on the enabled metric groups, a list of enabled metrics is created and sent in the form of `ConfigMap` to metric export agents. Metric export agents collect and export these enabled metrics, instead of all metrics on the system.

- The `MetricGroup` reconciler builds a `ConfigMap` for each type of metric (`vrouter` or `controller`) from the enabled `MetricGroup`(s) and applies the `ConfigMap` to all clusters.
- The KubeManager reconciler does the same for a new cluster.

Telemetry Exporter combines metric specifications with this `ConfigMap` to create enabled metric specifications. The metric export function only exports metrics from the enabled metric specifications, instead of all metrics.

Following items list YAML values for `ConfigMap` and `MetricGroup`.

ConfigMap: vrouter-export-enabled-metrics

- Revision number.
- Array of enabled metric names.

Custom Resource: MetricGroup

- Type: vrouter or controller
- Name: String
- Export: Boolean
- Metrics: Array of strings (metric name)

Install and Upgrade

`MetricGroup` is included in the analytics component in the CN2 Release 22.2. The predefined metric groups are automatically installed during the CN2 analytics deployment. See *Install Contrail Analytics for Upstream Kubernetes* or *Install Contrail Analytics for OpenShift Container Platform*.

Example: Predefined Metric Group

```
Bgpaas
Controller-bgp
Controller-info
Controller-peer
Controller-xmpp
Ermvpn
Evpn
Ipv4
Ipv6
```

```
Mvpn
Vrouter-cpu
Vrouter-info
Vrotuer-inv6
Vrouter-mem
Vrouter-traffic
Vrouter vmi
```

Example predefined `MetricGroup`: `vrouter-cpu` YAML file:

```
apiVersion: telemetry.juniper.net/v1alpha1
kind: MetricGroup
metadata:
  name: vrouter-cpu
  namespace: contrail-analytics
spec:
  export: true
  metricType: VROUTER
  metrics:
  - virtual_router_cpu_1min_load_avg
  - virtual_router_cpu_5min_load_avg
  - virtual_router_cpu_15min_load_avg
```

Manage MetricGroup with Kubectl Commands

You (the administrator) can manage `MetricGroup` with `kubectl` commands. Examples follow.

To delete `MetricGroup`:

```
kubectl delete metricgroup ipv6 -n contrail-analytics
```

To apply `MetricGroup`:

```
kubectl apply -f <yaml file with metric group definition>
```

To view `MetricGroup` resource:

```
kubectl get metricgroup ipv4 -n contrail-analytics -oyaml
```

To verify the existence following `ConfigMap`(s). Each cluster has its own copy of these two `ConfigMap`(s).

```
kubectl get cm -n contrail

Names of ConfigMap
    controller-export-enabled-metrics
    vrouter-export-enabled-metrics
```

Manage Metric Groups with UI

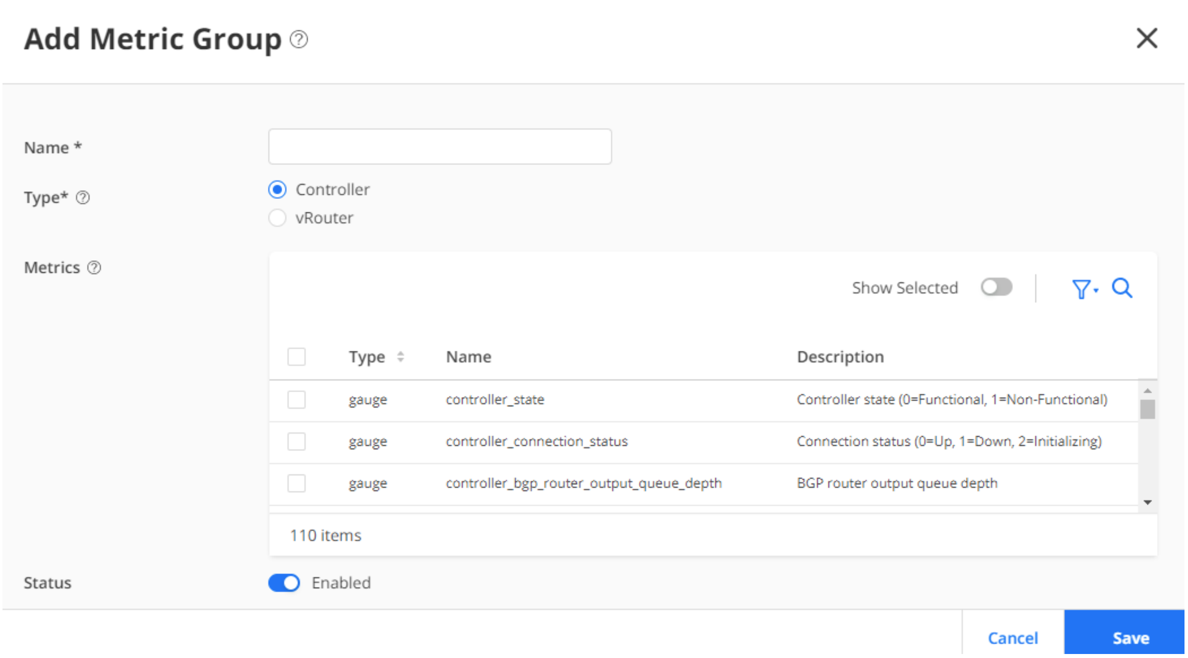
With this Tech Preview, you can manage `MetricGroups` using the CN2 Manager UI.
To manage Metric Groups in the UI:

1. Access the CN2 Manager UI from your browser:

   `https://`<**cluster-ip-address**>`/`
2. Log in to CN2 Manager by either:
   - Browse and select a `kubeconfig` file to upload.
   - Log in using a token.
3. From the left-navigation menu, select **Configure** > **Metric Groups**.
   The Metric Groups window displays.
4. To add a Metric Group, click the "**+**" icon in the upper right.

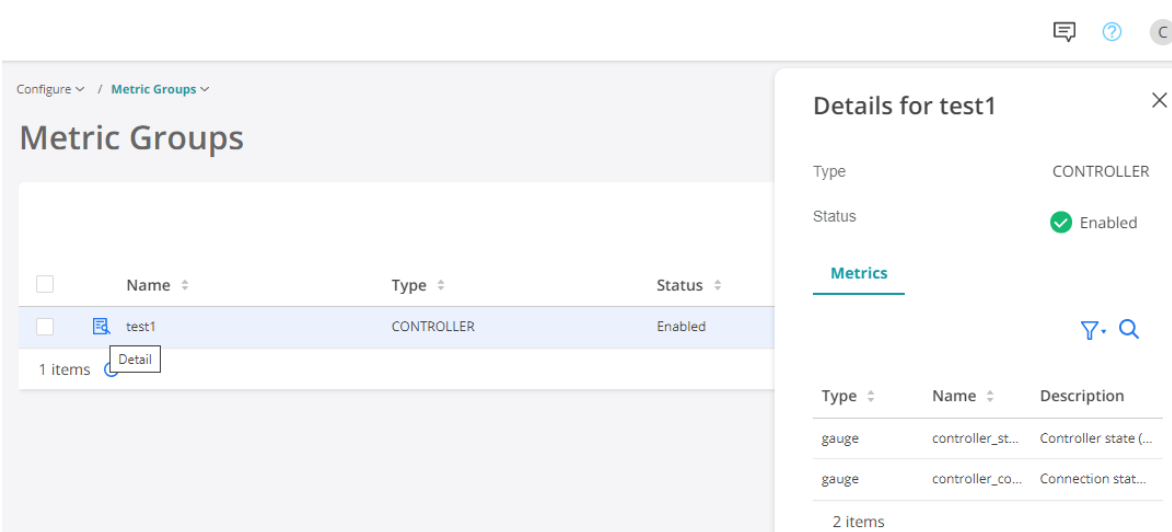
   Add the **Name**, select **Type**, and metrics to apply. Click **Save**.

Figure 12: Add a Metric Group



5. Click the detail icon to display the Metric Group you added.

Figure 13: Display Metric Group Detail



Juniper CN2 Technology Previews (Tech Previews)

Tech Previews give you the ability 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.