

Contrail® Networking

Contrail Networking Deployment Guide

Published
2023-09-26

RELEASE
1912

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

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Contrail® Networking Contrail Networking Deployment Guide
1912

Copyright © 2023 Juniper Networks, Inc. All rights reserved.

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

YEAR 2000 NOTICE

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

END USER LICENSE AGREEMENT

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

Table of Contents

[About This Guide | vi](#)

1

[Deploying Contrail Networking with Kubernetes](#)

[Contrail Networking with Kubernetes | 2](#)

[Contrail Integration with Kubernetes | 2](#)

[Kubernetes Updates | 9](#)

[Implementation of Kubernetes Network Policy with Contrail Firewall Policy | 13](#)

[Multiple Network Interfaces for Containers | 27](#)

2

[Deploying Contrail Networking with VMware vCenter](#)

[VMware vCenter with Containerized Contrail Networking | 33](#)

[Managing Networks From Contrail Command and VMware vCenter User Interfaces | 33](#)

[User Interfaces for Contrail Integration with VMware vCenter | 33](#)

[Feature Configuration for Contrail vCenter | 34](#)

[Create Virtual Network | 35](#)

[Delete Virtual Network | 37](#)

[Create a Virtual Machine – vCenter UI | 38](#)

[View Virtual Machine | 42](#)

[Integrating Contrail Release 5.0.X with VMware vRealize Orchestrator | 42](#)

3

[Deploying Contrail Networking with OpenStack](#)

[Configuring Virtual Networks | 49](#)

[Creating Projects in OpenStack for Configuring Tenants in Contrail | 49](#)

[Creating a Virtual Network with OpenStack Contrail | 51](#)

[Creating an Image for a Project in OpenStack Contrail | 55](#)

[Using Security Groups with Virtual Machines \(Instances\) | 58](#)

[Security Groups Overview | 58](#)

[Creating Security Groups and Adding Rules | 59](#)

[Using Contrail Resources in Heat Templates | 64](#)

Using the Contrail Heat Template | 64

QoS Support in Contrail Networking | 69

Quality of Service in Contrail | 69

Configuring Network QoS Parameters | 77

Overview | 77

QoS Configuration Examples | 78

Limitations | 79

Load Balancers | 80

Using Load Balancers in Contrail | 80

Support for OpenStack LBaaS Version 2.0 APIs | 94

Configuring Load Balancing as a Service in Contrail | 97

Overview: Load Balancing as a Service | 97

Contrail LBaaS Implementation | 98

Configuring LBaaS Using CLI | 99

Configuring LBaaS using the Contrail Web UI | 101

Optimizing Contrail Networking | 112

Multiqueue Virtio Interfaces in Virtual Machines | 112

Contrail Networking OpenStack Analytics | 114

Ceilometer Support in Contrail | 114

Overview | 114

Ceilometer Details | 115

Verification of Ceilometer Operation | 115

Contrail Ceilometer Plugin | 118

Ceilometer Installation and Provisioning | 121

Contrail OpenStack APIs | 122

Working with Neutron | 122

Data Structure | 122

Network Sharing in Neutron | 123

Commands for Neutron Network Sharing | 124

Support for Neutron APIs | 124

Contrail Neutron Plugin | 125

DHCP Options | 125

| Incompatibilities | 126

Deploying Contrail Networking on the Cloud

Multicloud Contrail Networking | 128

Deploying Contrail Multicloud with Contrail Command | 128

| Deploying Microsoft Azure with Contrail Command | 130

| Deploying Amazon Web Services with Contrail Command | 135

| Deploying Google Cloud Platform (GCP) with Contrail Command | 137

Adding a Compute Host to Multicloud | 142

Updating the Contrail Multicloud Cluster | 142

Deleting the Contrail MultiCloud Cluster | 145

Deploying Contrail Multicloud using REST API | 148

About This Guide

Use this guide to understand the next steps to be taken after successful installation of Contrail Networking. This guide describes the solutions available and the workflows involved in the deployment of Contrail Networking using the following orchestrators in the following configuration scenarios:

- Deploying Contrail Networking with Kubernetes
- Deploying Contrail Networking with VMware vCenter
- Deploying Contrail Networking with OpenStack
- Deploying Contrail Networking on the Public Cloud

Contrail Networking product documentation is organized into multiple guides as shown in [Table 1 on page vi](#), according to the task you want to perform or the deployment scenario.

Table 1: Contrail Networking Guides

Guide Name	Description
Contrail Networking Installation and Upgrade Guide	Provides step-by-step instructions to install and bring up Contrail and its various components.
Contrail Networking Deployment Guide	Provides information about the next steps to be taken after a successful installation of Contrail.
Contrail Networking Fabric Lifecycle Management Guide	Provides information about Contrail underlay management and data center automation.
Contrail Networking and Security User Guide	Provides information about creating and orchestrating highly secure virtual networks.
Contrail Networking Service Provider Focused Features Guide	Provides information about the features that are used by service providers.
Contrail Networking Analytics and Troubleshooting Guide	Provides information about AppFormix and Contrail analytics.

RELATED DOCUMENTATION

[README Access to Contrail Networking Registry 19XX](#)

[Contrail Networking Release Notes 1912](#)

[Contrail Networking Configuration API Reference, Release 1912](#)

[Tungsten Fabric Architecture Guide](#)

[Juniper Networks TechWiki: Contrail Networking](#)

1

PART

Deploying Contrail Networking with Kubernetes

[Contrail Networking with Kubernetes](#) | 2

Contrail Networking with Kubernetes

IN THIS CHAPTER

- [Contrail Integration with Kubernetes | 2](#)
- [Kubernetes Updates | 9](#)
- [Implementation of Kubernetes Network Policy with Contrail Firewall Policy | 13](#)
- [Multiple Network Interfaces for Containers | 27](#)

Contrail Integration with Kubernetes

IN THIS SECTION

- [What is Kubernetes? | 2](#)
- [Configuration Modes for Contrail Integrated with Kubernetes | 3](#)
- [Kubernetes Services | 6](#)
- [Ingress | 7](#)
- [Contrail Kubernetes Solution | 7](#)

Contrail Networking supports the Container Network Interface (CNI) for integrating Contrail with the Kubernetes automation platform.

What is Kubernetes?

Kubernetes, also called K8s, is an open source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure. It provides a portable platform across public and private clouds. Kubernetes supports deployment, scaling, and auto-healing of applications.

Kubernetes supports a pluggable framework called Container Network Interface (CNI) for most of the basic network connectivity, including container pod addressing, network isolation, policy-based security, a gateway, SNAT, load-balancer, and service chaining capability for Kubernetes orchestration. Contrail Release 4.0 provides support for CNI for Kubernetes.

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

[Table 2 on page 3](#) lists the mapping between Kubernetes concepts and OpenContrail resources.

Table 2: Kubernetes to OpenContrail Mapping

Kubernetes	OpenContrail Resources
Namespace	Shared or single project
Pod	Virtual-machine, Interface, Instance-ip
Service	ECMP-based native Loadbalancer
Ingress	HAProxy-based L7 Loadbalancer for URL routing
Network policy	Security group based on namespace and pod selectors

What is a Kubernetes Pod?

A Kubernetes pod is a group of one or more containers (such as Docker containers), the shared storage for those containers, and options on how to run the containers. Pods are always co-located and co-scheduled, and run in a shared context. The shared context of a pod is a set of Linux namespaces, cgroups, and other facets of isolation. Within the context of a pod, individual applications might have further sub-isolations applied.

You can find more information about Kubernetes at: <http://kubernetes.io/docs/whatisk8s/>.

Configuration Modes for Contrail Integrated with Kubernetes

Contrail can be configured in several different modes in Kubernetes. This section describes the various configuration modes.

Default Mode

In Kubernetes, all pods can communicate with all other pods without using network address translation (NAT). This is the default mode of Contrail Kubernetes cluster. In the default mode, Contrail creates a virtual-network that is shared by all namespaces, from which service and pod IP addresses are allocated.

All pods in all namespaces that are spawned in the Kubernetes cluster are able to communicate with one another. The IP addresses for all of the pods are allocated from a pod subnet that is configured in the Contrail Kubernetes manager.

NOTE: System pods that are spawned in the kube-system namespace are not run in the Kubernetes cluster; they run in the underlay, and networking for these pods is not handled by Contrail.

Namespace Isolation Mode

In addition to the default networking model mandated by Kubernetes, Contrail supports additional custom networking models that make available the many rich features of Contrail to the users of the Kubernetes cluster. One such feature is network isolation for Kubernetes namespaces.

For namespace isolation mode, the cluster administrator can configure a namespace annotation to turn on isolation. As a result, services in that namespace are not accessible from other namespaces, unless security groups or network policies are explicitly defined to allow access.

A Kubernetes namespace can be configured as isolated by annotating the Kubernetes namespace metadata:

```
opencontrail.org/isolation : true
```

Namespace isolation provides network isolation to pods, because the pods in isolated namespaces are not reachable to pods in other namespaces in the cluster.

Namespace isolation also provides service isolation to pods. If any Kubernetes service is implemented by pods in an isolated namespace, those pods are reachable only to pods in the same namespace through the Kubernetes service-ip.

To make services remain reachable to other namespaces, service isolation can be disabled by the following additional annotation on the namespace:

```
opencontrail.org/isolation.service : false
```

Disabling service isolation makes the services reachable to pods in other namespaces, however pods in isolated namespaces still remain unreachable to pods in other namespaces.

A namespace annotated as “isolated” for both pod and service isolation has the following network behavior:

- All pods created in an isolated namespace have network reachability with each other.
- Pods in other namespaces in the Kubernetes cluster *cannot* reach pods in the isolated namespace.
- Pods created in isolated namespaces *can* reach pods in non-isolated namespaces.
- Pods in isolated namespaces *can* reach non-isolated services in any namespace in the Kubernetes cluster.
- Pods from other namespaces *cannot* reach services in isolated namespaces.

A namespace annotated as “isolated”, with service-isolation disabled and only pod isolation enabled, has the following network behavior:

- All pods created in an isolated namespace have network reachability with each other.
- Pods in other namespaces in the Kubernetes cluster *cannot* reach pods in the isolated namespace.
- Pods created in isolated namespaces *can* reach pods in other namespaces.
- Pods in isolated namespaces *can* reach non-isolated services in any namespace in the Kubernetes cluster.
- Pods from other namespaces *can* reach services in isolated namespaces.

Custom Isolation Mode

Administrators and application developers can add annotations to specify the virtual network in which a pod or all pods in a namespace are to be provisioned. The annotation to specify this custom virtual network is:

```
"opencontrail.org/network: <fq_network_name>"
```

If this annotation is configured on a pod spec then the pod is launched in that network. If the annotation is configured in the namespace spec then all the pods in the namespace are launched in the provided network.

NOTE: The virtual network must be created using Contrail VNC APIs or Contrail-UI prior to configuring it in the pod or namespace spec.

Nested Mode

Contrail supports the provisioning of Kubernetes cluster inside an OpenStack cluster. While this nesting of clusters by itself is not unique, Contrail provides a *collapsed* control and data plane in which a single Contrail control plane and a single network stack manage and service both the OpenStack and Kubernetes clusters. With unified control and data planes, interworking and configuring these clusters is seamless, and the lack of replication and duplicity makes this a very efficient option.

In nested mode, a Kubernetes cluster is provisioned in the virtual machine of an OpenStack cluster. The CNI-plugin and the Contrail-kubernetes manager of the Kubernetes cluster interface directly with Contrail components that manage the OpenStack cluster.

In a nested-mode deployment, all Kubernetes features, functions, and specifications are supported as is. Nested deployment stretches the boundaries and limits of Kubernetes by allowing it to operate on the same plane as underlying OpenStack cluster.

For more information, see *Provisioning of Kubernetes Clusters*.

Kubernetes Services

A Kubernetes service is an abstraction that defines a logical set of pods and the policy used to access the pods. The set of pods implementing a service are selected based on the **LabelSelector** field in the service definition. In Contrail, a Kubernetes service is implemented as an ECMP-native load-balancer.

The Contrail Kubernetes integration supports the following **ServiceTypes**:

- **`clusterIP`**: This is the default mode. Choosing this **ServiceType** makes the service reachable through the cluster network.
- **`LoadBalancer`**: Designating a **ServiceType** as **`LoadBalancer`** enables the service to be accessed externally. The **`LoadBalancer`_Service_** is assigned both ClusterIP and ExternalIP addresses. This **ServiceType** assumes that the user has configured the public network with a floating-ip pool.

Contrail Kubernetes Service-integration supports TCP and UDP for protocols. Also, Service can expose more than one port where port and targetPort are different. For example:

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
```

```

- name: http
  protocol: TCP
  port: 80
  targetPort: 9376
- name: https
  protocol: TCP
  port: 443
  targetPort: 9377

```

Kubernetes users can specify `spec.clusterIP` and `spec.externalIPs` for both **LoadBalancer** and **clusterIP ServiceTypes**.

If **ServiceType** is **LoadBalancer** and no `spec.externalIP` is specified by the user, then contrail-kube-manager allocates a floating-ip from the public pool and associates it to the ExternalIP address.

Ingress

Kubernetes services can be exposed externally or exposed outside of the cluster in many ways. See <https://kubernetes.io/docs/concepts/services-networking/ingress/#alternatives> for a list of all methods of exposing Kubernetes services externally. Ingress is one such method. Ingress provides Layer 7 load-balancing whereas the other methods provide Layer 4 load-balancing. Contrail supports http-based single-service ingress, simple-fanout ingress, and name-based virtual hosting ingress.

Contrail Kubernetes Solution

Contrail Kubernetes solution includes the following elements.

Contrail Kubernetes Manager

The Contrail Kubernetes implementation requires listening to the Kubernetes API messages and creating corresponding resources in the Contrail API database.

A new module, `contrail-kube-manager`, runs in a Docker container to listen to the messages from the Kubernetes API server.

ECMP Load-Balancers for Kubernetes Services

Each service in Kubernetes is represented by a load-balancer object. The service IP allocated by Kubernetes is used as the VIP for the load-balancer. Listeners are created for the port on which the service is listening. Each pod is added as a member of the listener pool. The `contrail-kube-manager` listens for any changes based on service labels or pod labels, and updates the member pool list with any added, updated, or deleted pods.

Load-balancing for services is Layer 4 native, non-proxy load-balancing based on ECMP. The instance-ip (service-ip) is linked to the ports of each of the pods in the service. This creates an ECMP next-hop in Contrail and traffic is load-balanced directly from the source pod.

HAProxy Loadbalancer for Kubernetes Ingress

Kubernetes Ingress is implemented through the HAProxy load-balancer feature in Contrail. Whenever ingress is configured in Kubernetes, contrail-kube-manager creates the load-balancer object in contrail-controller. The Contrail service monitor listens for the load-balancer objects and launches the HAProxy with appropriate configuration, based on the ingress specification rules in active-standby mode.

See "[Using Load Balancers in Contrail](#)" on page 80 for more information on load balancers.

Security Groups for Kubernetes Network Policy

Kubernetes network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints. **NetworkPolicy** resources use labels to select pods and define white list rules which allow traffic to the selected pods in addition to what is allowed by the isolation policy for a given namespace.

For more information about Kubernetes network policies, see <https://kubernetes.io/docs/concepts/services-networking/networkpolicies/>.

The contrail-kube-manager listens to the Kubernetes network policy events for create, update, and delete, and translates the Kubernetes network policy to Contrail security group objects applied to virtual machine interfaces (VMIs). The VMIs are dynamically updated as pods and labels are added and deleted.

Kubernetes Support for Security Policy

Network policies created in a Kubernetes environment are implemented by using Contrail Security Policy framework. Labels from the Kubernetes environment are exposed as tags in Contrail. Starting in Contrail Release 5.0, you can define tags for a Kubernetes environment. Contrail security policy uses these tags to implement specified Kubernetes policies. You can define tags in the UI or upload configurations in JSON format. The newly-defined tags can be used to create and enforce policies in Contrail Security.

Domain Name Server (DNS)

Kubernetes implements DNS using SkyDNS, a small DNS application that responds to DNS requests for service name resolution from pods. SkyDNS runs as a pod in Kubernetes.

Supported Kubernetes Annotations

Currently, Contrail Networking supports the following Kubernetes annotations:

```
'opencontrail.org/network': '{"domain": "default-domain", "project": "k8s-contrail",
"name": "deu"}'
'opencontrail.org/isolation': 'true'
'opencontrail.org/fip-pool': '{"domain": "default-domain", "project": "k8s-default", "network":
"k8s-default-svc-public", "name": "default"}'
```

For further details, refer to <https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/>.

RELATED DOCUMENTATION

| *Verifying Configuration for CNI for Kubernetes*

Kubernetes Updates

IN THIS SECTION

- [TLS Bootstrapping of Kubernetes Nodes | 10](#)
- [Priority Based Multitenancy | 10](#)
- [Improved Autoscaling | 10](#)
- [Reachability to Kubernetes Pods Using the IP Fabric Forwarding Feature | 10](#)
- [Service Isolation Through Virtual Networks | 10](#)
- [Contrail ip-fabric-snat Feature | 11](#)
- [Third-Party Ingress Controllers | 11](#)
- [Custom Network Support for Ingress Resources | 11](#)
- [Kubernetes Probes and Kubernetes Service Node-Port | 12](#)
- [Kubernetes Network-Policy Support | 12](#)

This topic describes updates to Kubernetes and supported features in Contrail.

TLS Bootstrapping of Kubernetes Nodes

Contrail Release 5.1 supports TLS Bootstrapping of Kubernetes Nodes. TLS bootstrapping streamlines Kubernetes' ability to add and remove nodes from the Contrail cluster.

Priority Based Multitenancy

Contrail Release 5.1 supports priority on the various resource quotas through the ResourceQuotaScopeSelector feature.

Improved Autoscaling

Contrail release 5.1 supports improved pod autoscaling by creating and deleting pods based on the load.

Reachability to Kubernetes Pods Using the IP Fabric Forwarding Feature

A Kubernetes pod is a group of one or more containers (such as Docker containers), the shared storage for those containers, and options on how to run the containers. Since pods are in the overlay network, they cannot be reached directly from the underlay without a gateway or vRouter. In Contrail Release 5.0, the IP fabric forwarding (ip-fabric-forwarding) feature enables virtual networks to be created as part of the underlay network and eliminates the need for encapsulation and decapsulation of data. The ip-fabric-forwarding feature is only applicable for pod networks. If ip-fabric-forwarding is enabled, pod-networks are associated to ip-fabric-ipam instead of pod-ipam which is also a flat subnet.

The ip-fabric-forwarding feature is enabled and disabled in the global and namespace levels. By default, ip-fabric-forwarding is disabled in the global level. To enable it in global level, you must set "ip_fabric_forwarding" to "true" in the "[KUBERNETES]" section of the `/etc/contrail/contrail-kubernetes.conf` file. To enable or disable the feature in namespace level, you must set "ip_fabric_forwarding" to "true" or "false" respectively in namespace annotation. For example, "opencontrail.org/ip_fabric_forwarding": "true". Once the feature is enabled, it cannot be disabled.

For more information, see [Gateway-less Forwarding](#).

Service Isolation Through Virtual Networks

In namespace isolation mode, services in one namespace are not accessible from other namespaces, unless security groups or network policies are explicitly defined to allow access. If any Kubernetes service is implemented by pods in an isolated namespace, those services are reachable only to pods in the same namespace through the Kubernetes service-ip.

The Kubernetes service-ip is allocated from the cluster network despite being in an isolated namespace. So, by default, service from one namespace can reach services from another namespace. However, security groups in isolated namespaces prevent reachability from external namespace and also prevent reachability from outside of the cluster. In order to enable access by external namespaces, the security group must be edited to allow access to all namespaces which defeats the purpose of isolation.

Contrail Release 5.0 enables service or ingress reachability from external clusters in isolated namespaces. Two virtual networks are created in isolated namespaces. One network is dedicated to pods and one is dedicated to services. Contrail network-policy is created between the pod network and the service network for reachability between pods and services. Service uses the same service-ipam which is a flat-subnet like pod-ipam. It is applicable for default namespace as well.

Contrail ip-fabric-snat Feature

Starting with Contrail Release 5.0, with the Contrail ip-fabric-snat feature, pods that are in the overlay can reach the Internet without floating IPs or a logical-router. The ip-fabric-snat feature uses compute node IP for creating a source NAT to reach the required services and is applicable only to pod networks. The kube-manager reserves ports 56000 through 57023 for TCP and 57024 through 58047 for UDP to create a source NAT in global-config during the initialization.

The ip-fabric-snat feature can be enabled or disabled in the global or namespace levels. By default, the feature is disabled in the global level. To enable the ip-fabric-snat feature in the global level, you must set “ip-fabric-snat” to “true” in the “[KUBERNETES]” section in the `/etc/contrail/contrail-kubernetes.conf` file. To enable or disable it in the namespace level, you must set “ip_fabric_snat” to “true” or “false” respectively in namespace annotation. For example, “opencontrail.org/ip_fabric_snat”: “true”. The ip_fabric_snat feature can be at enabled and disabled any time. To enable or disable the ip_fabric_snat feature in the default-pod-network, default namespace must be used. If the ip_fabric_forwarding is enabled, ip_fabric_snat is ignored.

For more information, see [Distributed SNAT](#).

Third-Party Ingress Controllers

Multiple ingress controllers can co-exist in Contrail. If “kubernetes.io/ingress.class” is absent or is “opencontrail” in the annotations of the Kubernetes ingress resource, the kube-manager creates a HAProxy loadbalancer. Otherwise it is ignored and the respective ingress controller handles the ingress resource. Since Contrail ensures the reachability between pods and services, any ingress controller can reach the endpoints or pods directly or through services.

Custom Network Support for Ingress Resources

Contrail supports custom networks in namespace level for pods. Starting with Contrail Release 5.0, custom networks are supported for ingress resources as well.

Kubernetes Probes and Kubernetes Service Node-Port

The Kubelet 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. So, any process in the node can reach the pods.

Kubernetes Service Node-Port is based on node reachability to pods. Since Contrail provides connectivity between node and pods through Contrail the network policy, Node Port is supported.

Kubernetes Network-Policy Support

Contrail Release 5.1 supports the following Kubernetes release 1.12 network policy features:

- Egress support for network policy—Each NetworkPolicy includes a policyTypes list which can include either Ingress, Egress, or both. The policyTypes field indicates whether or not the given policy applies to ingress traffic to selected pod, egress traffic from the selected pod, or both. Contrail Release 5.1 supports the podSelector&namespaceSelector egress specification. Contrail Release 5.0 supports podSelector, namespaceSelector, and egress CIDR egress specifications.
- Classless Interdomain Routing (CIDR) selector support for egress and ingress network policies
- Contrail-ansible-deployer provisioning—Contrail-ansible-deployer is updated to support Kubernetes 1.12.

Contrail Release 5.0 supports Kubernetes release 1.9.2 and enables implementing Kubernetes network policy in Contrail using the Contrail firewall security policy framework. While Kubernetes network policy can be implemented using other security objects in Contrail like security groups and Contrail network policies, the support of tags by Contrail firewall security policy aids in the simplification and abstraction of workloads.

For more information, see *Implementation of Kubernetes Network Policy with Contrail Firewall Policy*.

RELATED DOCUMENTATION

Implementation of Kubernetes Network Policy with Contrail Firewall Policy

Contrail Integration with Kubernetes

Implementation of Kubernetes Network Policy with Contrail Firewall Policy

IN THIS SECTION

- [Kubernetes Network Policy Characteristics | 13](#)
- [Representing Kubernetes Network Policy as Contrail Firewall Security Policy | 14](#)
- [Contrail Firewall Policy Naming Convention | 16](#)
- [Implementation of Kubernetes Network Policy | 17](#)
- [Example Network Policy Configurations | 17](#)
- [Cluster-wide Policy Action Enforcement | 25](#)

Contrail Release 5.0 supports implementing Kubernetes 1.9.2 network policy in Contrail using the Contrail firewall security policy framework. While Kubernetes network policy can be implemented using other security objects in Contrail like security groups and Contrail network policies, the support of tags by Contrail firewall security policy aids in the simplification and abstraction of Kubernetes workloads.

Contrail firewall security policy allows decoupling of routing from security policies and provides multi-dimension segmentation and policy portability while significantly enhancing user visibility and analytics functions. Contrail firewall security policy uses tags to achieve multi-dimension traffic segmentation among various entities, and with security features. Tags are key-value pairs associated with different entities in the deployment. Tags can be pre-defined or custom defined. Kubernetes network policy is a specification of how groups of Kubernetes workloads, which are hereafter referred to as pods, are allowed to communicate with each other and other network endpoints. Network policy resources use labels to select pods and define rules which specify what traffic is allowed to the selected pods.

Kubernetes Network Policy Characteristics

Kubernetes network policies have the following characteristics:

- A network policy is pod specific and applies to a pod or a group of pods. If a specified network policy applies to a pod, the traffic to the pod is dictated by rules of the network policy.
- If a network policy is not applied to a pod then the pod accepts traffic from all sources.
- A network policy can define traffic rules for a pod at the ingress, egress, or both directions. By default, a network policy is applied to the ingress direction, if no direction is explicitly specified.

- When a network policy is applied to a pod, the policy must have explicit rules to specify a whitelist of permitted traffic in the ingress and egress directions. All traffic that does not match the whitelist rules are denied and dropped.
- Multiple network policies can be applied on any pod. Traffic matching any one of the network policies must be permitted.
- A network policy acts on connections rather than individual packets. For example, if traffic from pod A to pod B is allowed by the configured policy, then the return packets for that connection from pod B to pod A are also allowed, even if the policy in place does not allow pod B to initiate a connection to pod A.
- **Ingress Policy:** An ingress rule consists of the identity of the source and the protocol:port type of traffic from the source that is allowed to be forwarded to a pod.

The identity of the source can be of the following types:

- Classless Interdomain Routing (CIDR) block—If the source IP address is from the CIDR block and the traffic matches the protocol:port, then traffic is forwarded to the pod.
- Kubernetes namespaces—Namespace selectors identify namespaces, whose pods can send the defined protocol:port traffic to the ingress pod.
- Pods—Pod selectors identify the pods in the namespace corresponding to the network policy, that can send matching protocol:port traffic to the ingress pods.
- **Egress Policy:** This specifies a whitelist CIDR to which a particular protocol:port type of traffic is permitted from the pods targeted by the network policy

The identity of the destination can be of the following types:

- CIDR block—If the destination IP address is from the CIDR block and the traffic matches the protocol:port, then traffic is forwarded to the destination.
- Kubernetes namespaces—Namespace selectors identify namespaces, whose pods can send the defined protocol:port traffic to the egress pod.
- Pods—Pod selectors identify the pods in the namespace corresponding to the network policy, that can receive matching protocol:port traffic from the egress pods.

Representing Kubernetes Network Policy as Contrail Firewall Security Policy

Kubernetes and Contrail firewall policy are different in terms of the semantics in which network policy is specified in each. The key to efficient implementation of a Kubernetes network policy through Contrail firewall policy is in mapping the corresponding configuration constructs between these two entities.

The constructs are mapped as displayed in [Table 3 on page 15](#):

Table 3: Kubernetes Network Policy and Contrail Firewall Policy Mapping

Kubernetes Network Policy Constructs	Contrail Firewall Policy Constructs
Label	Custom Tag (one for each label)
Namespace	Custom Tag (one for each namespace)
Network Policy	Firewall Policy (one firewall policy per Network Policy)
Rule	Firewall Rule (one firewall rule per network policy rule)
CIDR Rules	Address Group
Cluster	Default Application Policy Set

NOTE: The project in which Contrail firewall policy constructs are created is the one that houses the Kubernetes cluster. For example, the Contrail firewall policy constructs are created in the global scope, if the Kubernetes cluster is a standalone cluster and the Contrail firewall policy constructs are created in the project scope, if the Kubernetes cluster is a nested cluster.

Resolving Kubernetes Network Policy Labels

The representation of pods in Contrail firewall policy is exactly the same as in the corresponding Kubernetes network policy. Contrail firewall policy deals with labels or tags in Contrail terminology. Contrail does not expand labels to IP addresses.

For example, in the default namespace, if network policy-podSelector specifies: role=db, then the corresponding firewall rule specifies the pods as (role=db && namespace=default). No other translations to pod IP address or otherwise are done.

If the same network-policy also has namespaceSelector as namespace=myproject, then the corresponding firewall rule represents that namespace as (namespace=myproject). No other translations or rules representing pods in “myproject” namespace is done.

Similarly, each CIDR is represented by one rule. In essence, the Kubernetes network policy is translated 1:1 to Contrail firewall policy. There is only one additional firewall rule created for each Kubernetes network policy. The purpose of that rule is to implement the implicit deny requirements of the network policy and no other rule is created.

Contrail Firewall Policy Naming Convention

Contrail firewall security policies and rules are named as follows:

- A Contrail firewall security policy created for a Kubernetes network policy is named in the following format:

```
< Namespace-name >-< Network Policy Name >
```

For example, a network policy "world" in namespace "Hello" is named:

```
Hello-world
```

- Contrail firewall rules created for a Kubernetes network policy are named in the following format:

```
< Namespace-name >-<PolicyType>-< Network Policy Name >-<Index of from/to blocks>-<selector type>-<rule-index>-<svc/port index>
```

For example:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: world
  namespace: hello
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
  ingress:
    - from:
      - podSelector:
          matchLabels:
            role: frontend
```

A rule corresponding to this policy is named:

```
hello-ingress-world-0-podSelector-0-0
```

Implementation of Kubernetes Network Policy

The contrail-kube-manager daemon binds Kubernetes and Contrail together. This daemon connects to the API server of Kubernetes clusters and coverts Kubernetes events, including network policy events, into appropriate Contrail objects. With respect to a Kubernetes network policy, contrail-kube-manager performs the following actions:

- Creates a Contrail tag for each Kubernetes label
- Creates a firewall policy for each Kubernetes network policy
- Creates an Application Policy Set (APS) to represent the cluster. All firewall policies created in that cluster are attached to this application policy set.
- Modifications to existing Kubernetes network policies result in the corresponding firewall policies being updated.

Example Network Policy Configurations

The following examples illustrate various sample network policies and the corresponding firewall security policies created.

Example 1 - Conditional egress and ingress traffic

The following policy specifies a sample network policy with specific conditions for ingress and egress traffic to and from all pods in a namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
```



```

policyTypes:
- Ingress
- Egress
ingress:
- from:
  - ipBlock:
      cidr: 17x.xx.0.0/16
      except:
        - 17x.xx.1.0/24
  - namespaceSelector:
      matchLabels:
        project: myproject
  - podSelector:
      matchLabels:
        role: frontend
  ports:
    - protocol: TCP
      port: 6379
egress:
- to:
  - ipBlock:
      cidr: 10.0.0.0/24
  ports:
    - protocol: TCP
      port: 5978

```

Sample Contrail firewall security policy

The test-network-policy defined in Kubernetes results in the following objects being created in Contrail.

Tags—The following tags are created, if they do not exist. In a regular workflow, these tags must have been created by the time the namespace and pods were created.

Key	Value
role	db
namespace	default

Address Groups

The following address groups are created:

Name	Prefix
17x.xx.1.0/24	17x.xx.1.0/24
17x.xx.0.0/16	17x.xx.0.0/16
10.0.0.0/24	10.0.0.0/24

Firewall Rules

The following firewall rules are created:

Rule Name	Action	Services	Endpoint1	Dir	Endpoint2	Match Tags
default-ingress-test-network-policy-0-ipBlock-0-17x.xx.1.0/24-0	deny	tcp:6379	Address Group: 17x.xx.1.0/24	>	role=db && namespace=default	
default-ingress-test-network-policy-0-ipBlock-0-cidr-17x.xx.0.0/16-0	pass	tcp:6379	Address Group: 17x.xx.0.0/16	>	role=db && namespace=default	
default-ingress-test-network-policy-0-namespaceSelector-1-0	pass	tcp:6379	project=myproject	>	role=db && namespace=default	
default-ingress-test-network-policy-0-podSelector-2-0	pass	tcp:6379	namespace=default && role=frontend	>	role=db && namespace=default	
default-egress-test-network-policy-ipBlock-0-cidr-10.0.0.0/24-0	pass	tcp:5978	role=db && namespace=default	>	Address Group: 10.0.0.0/24	

Firewall Policy

The following firewall security policy is created with the following rules.

Name	Rules
default-test-network-policy	<ul style="list-style-type: none"> • default-ingress-test-network-policy-0-ipBlock-0-17x.xx.1.0/24-0 • default-ingress-test-network-policy-0-ipBlock-0-cidr-17x.xx.0.0/16-0 • default-ingress-test-network-policy-0-namespaceSelector-1-0 • default-ingress-test-network-policy-0-podSelector-2-0 • default-egress-test-network-policy-ipBlock-0-cidr-10.0.0.0/24-0

Example 2 - Allow all Ingress Traffic

The following policy explicitly allows all traffic for all pods in a namespace:

Sample Kubernetes network policy

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector:
  ingress:
  - {}

```

Sample Contrail firewall security policy

Tags—The following tags are created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules

The following firewall rule is created:

Rule Name	Action	Services	Endpoint1	Dir	Endpoint2	Match Tags
default-ingress-allow-all-ingress-0-allow-all-0	pass	any	any	>	namespace=default	

Firewall Policy

The following firewall policy are created:

Name	Rules
default-allow-all-ingress	default-ingress-allow-all-ingress-0-allow-all-0

Example 3 - Deny all ingress traffic

The following policy explicitly denies all ingress traffic to all pods in a namespace:

Sample Kubernetes network policy

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-ingress
spec:
  podSelector:
  policyTypes:
    - Ingress

```

Sample Contrail firewall security policy

Tags—The following tags are created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules - None

NOTE: The implicit behavior of any network policy is to deny traffic not matching explicit allow flows. However in this policy, there are no explicit allow rules. Hence, no firewall rules are created for this policy.

Firewall Policy

The following firewall policy is created:

Name	Rules
default-deny-ingress	

Example 4 - Allow all egress traffic

The following policy explicitly allows all egress traffic from all pods in a namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress
spec:
  podSelector:
  egress:
  - {}
```

Sample Contrail firewall security policy

Tags—The following tag is created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules

The following firewall rule is created:

Rule Name	Action	Services	Endpoint1	Dir	Endpoint2	Match Tags
default-egress-allow-all-egress-allow-all-0	pass	any	namespace=default	>	any	

Firewall Policy

The following firewall policy is created:

Name	Rules
default-allow-all-egress	default-egress-allow-all-egress-allow-all-0

Example 5 - Default deny all egress traffic

The following policy explicitly denies all egress traffic from all pods in a namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-egress
spec:
  podSelector: {}
  policyTypes:
    - Egress
```

Sample Contrail firewall security policy

Tags—The following tag is created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules - None

NOTE: The implicit behavior of any network policy with egress policy type is to deny egress traffic not matching explicit egress allow flows. In this policy, there are no explicit egress allow rules. Hence, no firewall rules are created for this policy.

Firewall Policy

The following firewall policy is created:

Name	Rules
default-deny-all-egress	

Example 6 - Default deny all ingress and egress traffic

The following policy explicitly denies all ingress and egress traffic to and from all pods in that namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-ingress-egress
spec:
  podSelector:
  policyTypes:
    - Ingress
    - Egress
```

Sample Contrail firewall security policy

Tags—The following tags is created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules - None

NOTE: The implicit behavior of any network policy with ingress/egress policy type is to deny corresponding traffic not matching explicit allow flows. In this policy, there are no explicit allow rules. Hence, no firewall rules are created for this policy.

Firewall Policy

The following firewall policy is created:

Name	Rules
default-deny-all-ingress-egress	

Cluster-wide Policy Action Enforcement

The specification and the syntax of network policies allow for maximum flexibility and varied combinations. However, you must exercise caution while configuring the network policies.

Consider a case where two network policies are created:

- Policy 1: Pod A can send to Pod B.
- Policy 2: Pod B can only receive from Pod C.

From a networking flow perspective, there is an inherent contradiction between the above policies. Policy 1 states that a flow from Pod A to Pod B is allowed. Policy 2 implies that flow from Pod A to Pod B is not allowed. From a networking perspective, Contrail prioritizes flow behavior as more critical. In the event of inherent contradiction in network policies, Contrail will honor the flow perspective. One of the core aspects of this notion is that if a policy matches a flow, the action is honored cluster-wide.

For instance, if a flow matches a policy at the source, the flow will match the same policy in the destination as well. Therefore, the flow behavior in a Contrail-managed Kubernetes cluster is as shown below:

- Flow from Pod A to Pod B is allowed (due to Policy 1)
- Flow from Pod C to Pod B is allowed (due to Policy 2)
- Any other flow to Pod B is disallowed (due to Policy 2)

Example Network Policy Action Enforcement Scenarios

Consider the following examples of network policy action enforcement:

- Allow all egress traffic and deny all ingress traffic

Setup: Namespace NS1 has two pods, Pod A and Pod B.

Policy: A network policy applied on namespace NS1 states:

- Rule 1. Allow all egress traffic from all pods in NS1.
- Rule 2. Deny all ingress traffic to all pods in NS1.

Behavior:

- Pod A can send traffic to Pod B (due to rule 1)
- Pod B can send traffic to Pod A (due to rule 1)
- PodX from a different namespace cannot send traffic to Pod A or Pod B (due to rule 2)
- Allow all ingress traffic and deny all egress traffic

Setup: Namespace NS1 has two pods, Pod A and Pod B.

Policy: A network policy applied on namespace NS1 states:

- Rule 1. Allow all ingress traffic to all pods in NS1
- Rule 2. Deny all egress traffic from all pods in NS1.

Behavior:

- Pod A can send traffic to Pod B (due to rule 1)
- Pod B can send traffic to Pod A (due to rule 1)
- Pod A and Pod B cannot send traffic to pods in any other namespace.
- Egress CIDR rule

Setup: Namespace NS1 has two pods, Pod A and Pod B.

Policy: A network policy applied on namespace NS1 states:

- Policy 1: Allow Pod A to send traffic to CIDR of Pod B.
- Policy 2: Deny all ingress traffic to all pods in NS1.

Behavior:

- Pod A can send traffic to Pod B (due to Policy 1)
- All other traffic to Pod A and Pod B is dropped (due to policy 2)

RELATED DOCUMENTATION

| *Kubernetes Updates*

Multiple Network Interfaces for Containers

Starting in Release 4.0, Contrail provides networking support for containers using Kubernetes Orchestration. You can allocate a network interface to every container that you create using standard Container Networking Interface (CNI plugin). For more information on Contrail Containers Networking, see *Contrail Integration with Kubernetes*.

With Contrail Release 5.1, you can allocate multiple network interfaces (multi-net) to a container to enable the container to connect to multiple networks. You can specify the networks the container can connect to. A network interface is either a physical interface or a virtual interface and is connected to the Linux network namespace. A network namespace is the network stack in the Linux kernel. More than one container can share the same network namespace.

The following limitations and caveats apply when you create multi-net interfaces:

- You cannot add or remove sidecar networks while the pod is still running.
- The administrator is responsible for removing corresponding Contrail pods before deleting the network attachment definition from the Kubernetes API server.
- Contrail creates a default cluster-wide-network in addition to custom networks.
- Contrail CNI plugin is not a delegating plugin. It does not support specifications for delegating plugins that are provided in the Kubernetes Network Custom Resource Definition De Facto Standard Version 1. For more information, view [v1] **Kubernetes Network Custom Resource Definition De-facto Standard.md** from the <https://github.com/K8sNetworkPlumbingWG/multi-net-spec> page.

Contrail multi-net support is based on the Kubernetes multi-net model. Kubernetes multi-net model has a specific design and construct, and can be extended to non-kubernetes models like Contrail multi-net. Contrail multi-net model does not require changes to the Kubernetes API and Kubernetes CNI driver. Contrail multi-net model, as in the case of Kubernetes multi-net model, does not change the existing cluster-wide network behavior.

Creating Multi-Net Interfaces

Follow these steps to create multi-net interfaces.

1. Create Network Object Model.

You create the network object model if the cluster does not support the model.

The object model of the container orchestration platform represents the network and attaches the network to a container. If the model does not support network objects by default, you can use extensions to represent the network.

Creating Network Object Model by using Kubernetes NetworkAttachmentDefinition CRD object

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the form: <plural>.<group>
  name: network-attachment-definitions.k8s.cni.cncf.io
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: k8s.cni.cncf.io
  # version name to use for REST API: /apis/<group>/<version>
  version: v1
  # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: network-attachment-definitions
    # singular name to be used as an alias on the CLI and for display
    singular: network-attachment-definition
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: NetworkAttachmentDefinition
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
      - net-attach-def
  validation:
    openAPIV3Schema:
      properties:
```

```
spec:
  properties:
    config:
      type: string
```

Kubernetes uses custom extensions to represent networks in its object model. CustomResourceDefinition(CRD) feature of Kubernetes helps support custom extensions.

NOTE: A CRD is created automatically when you install Contrail. Networks specified by CRD are sidecars that are not recognized by Kubernetes. The interaction of additional pod network attachments with Kubernetes API and its objects, such as services, endpoints, proxies, etc. are not specified. Kubernetes does not recognize the association of these objects to any pod.

2. Create networks.

You create networks in the cluster:

- Through the API server.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    opencontrail.org/cidr: "<ip address>/24"
    opencontrail.org/ip_fabric_forwarding: "false"
    opencontrail.org/ip_fabric_snat: "false"
  name: right-network
  namespace: default
spec:
  config: '{ "cniVersion": "0.3.0", "type": "contrail-k8s-cni" }'
```

Create a **right-network.yaml** file.

- By mapping to an existing network created from the Contrail Web user interface or from the Contrail Command user interface.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: extns-network
  annotations:
```

```

    "opencontrail.org/network" : '{"domain":"default-domain", "project": "k8s-extns",
    "name":"k8s-extns-pod-network"}'
  spec:
    config: '{
      "cniVersion": "0.3.1",
      "type": "contrail-k8s-cni"
    }'

```

Command to create the network:

```
kubectl apply -f right-network.yaml
```

3. Assign networks to pods.

You assign the networks that you created in Step 2 to pods. Each pod also has a default network assigned to it. Therefore, each pod will have the following networks assigned:

- default network (assigned by Kubernetes)

NOTE: Contrail internally creates a default network called `cluster-wide-network`. This interface is the default interface for the pod

- network that you created in Step 2

Assigning networks to pods by using *k8s-semantics*:

Option 1

```

apiVersion: v1
kind: Pod
metadata:
  name: multiNetworkPod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      { "name": "network-a" },
      { "name": "network-b" }
    ]'
spec:
  containers:
    - image: busybox
      command:
        - sleep

```

```

    - "3600"
    imagePullPolicy: IfNotPresent
    name: busybox
    stdin: true
    tty: true
    restartPolicy: Always

```

Option 2

```

apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-pod-3
  annotations:
    k8s.v1.cni.cncf.io/networks: left-network,blue-network,right-network,extns/data-network
spec:
  containers:
    - name: ubuntuapp
      image: ubuntu-upstart
      securityContext:
        capabilities:
          add:
            - NET_ADMIN

```

RELATED DOCUMENTATION

| *Contrail Integration with Kubernetes*

2

PART

Deploying Contrail Networking with VMware vCenter

VMware vCenter with Containerized Contrail Networking | 33

CHAPTER 2

VMware vCenter with Containerized Contrail Networking

IN THIS CHAPTER

- [Managing Networks From Contrail Command and VMware vCenter User Interfaces | 33](#)
- [Integrating Contrail Release 5.0.X with VMware vRealize Orchestrator | 42](#)

Managing Networks From Contrail Command and VMware vCenter User Interfaces

IN THIS SECTION

- [User Interfaces for Contrail Integration with VMware vCenter | 33](#)
- [Feature Configuration for Contrail vCenter | 34](#)

You can install Contrail to work with the VMware vCenter Server version 6.5. You use the Contrail Command user interface (UI) and the vCenter UI to configure and manage the integrated Contrail system.

These topics provide instructions on how to use the Contrail Command UI and the vCenter UI to configure and manage features of a Contrail VMware integrated system.

User Interfaces for Contrail Integration with VMware vCenter

The Contrail Command and VMware vCenter UIs are available after installing the integrated Contrail system.

After Contrail is integrated with VMware vCenter, you can use the UIs to manage and configure features of the system.

Contrail Administration User Interface

Contrail Command is an administrator’s user interface. It provides a view of all components managed by the Contrail controller.

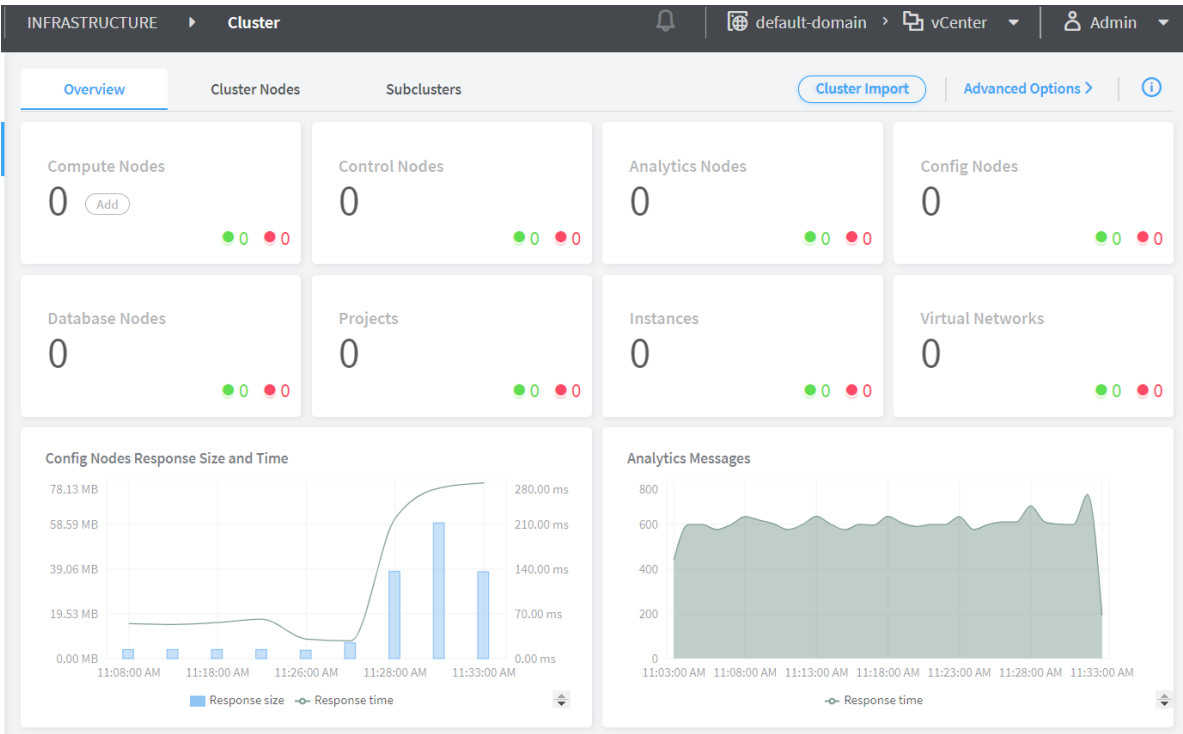
1. To log in to Contrail Command, use your Contrail server main IP address URL as follows:

```
https://<contrail command ip>:9091
```

2. Then log in using your registered Contrail account administrator credentials.

Upon successful login, the Contrail vCenter UI is displayed. See [Figure 1 on page 34](#).

Figure 1: Contrail vCenter User Interface



Feature Configuration for Contrail vCenter

IN THIS SECTION

- Create Virtual Network | 35
- Delete Virtual Network | 37
- Create a Virtual Machine – vCenter UI | 38

This section provides instructions on how to create, delete, and view virtual networks by using the Contrail Command UI. This section also provides instructions on how to create a virtual machine by using the vCenter client.

Create Virtual Network

Follow these steps to create a virtual network by using the Contrail Command UI.

1. Click **Overlay** > **Virtual Networks** to access the All Networks page.

NOTE: Ensure that you are in the vCenter project.

2. Click **Create**.

The Network tab of the Create Virtual Network page is displayed.

3. Enter the name of the virtual network in the **Name** field.
4. Select a network policy from the **Network Policies** list.
5. Select an allocation mode from the **Allocation Mode** list.
6. Click **+ Add** in the Subnets section and add the following information in the fields that are displayed.
 - a. Select a network IPAM service from the **Network IPAM** list.
 - b. Enter valid IPv4 subnet/mask in the **CIDR** field.
 - c. Enter allocation pool information in the **Allocation Pools** field.
 - d. Enter service IP address in the **Service Address** field.
 - e. Click **Create** to add subnet information.

[Figure 2 on page 36](#) shows the creation of a virtual network called Green-VN.

Figure 2: Create a Virtual Network

The screenshot shows the 'Create Virtual Network' form in a vCenter interface. The form is divided into three tabs: Network, Tags, and Permissions. The 'Network' tab is active. The form contains the following fields and options:

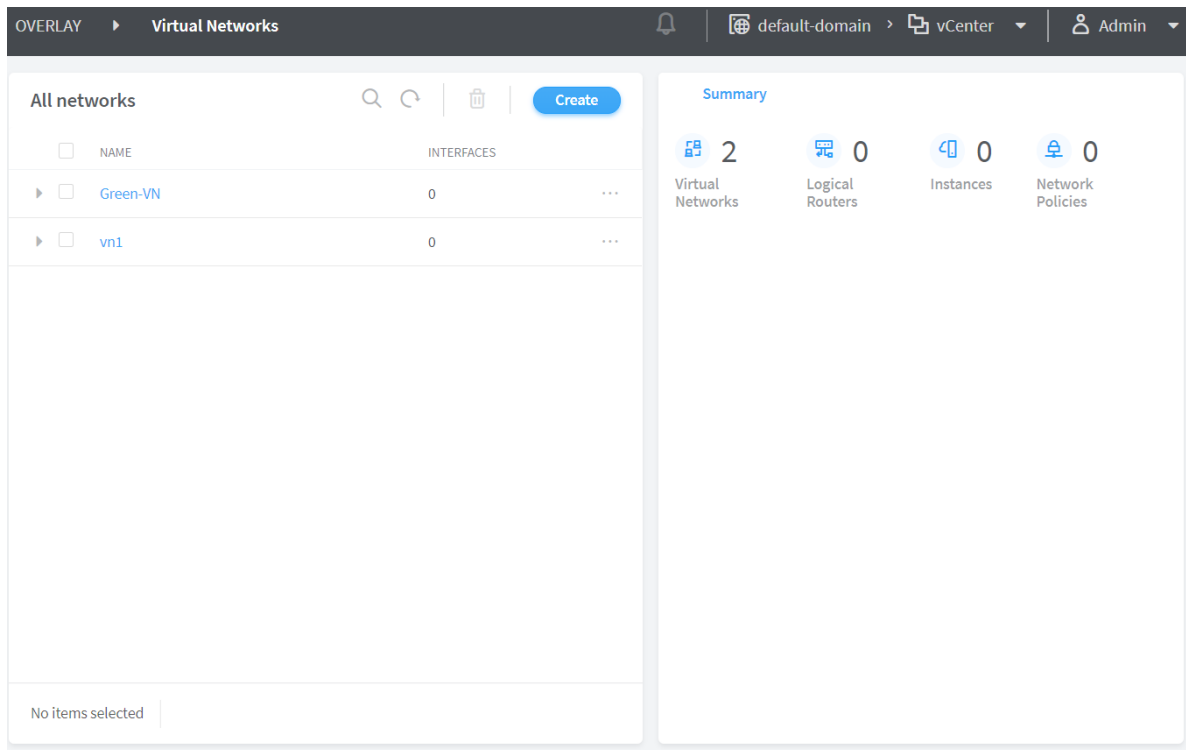
- Name***: A text input field containing 'Green-VN'.
- Network Policies**: A dropdown menu with 'Select Network Policies'.
- Allocation Mode**: A dropdown menu with 'User defined subnet only'.
- VxLAN Network Identifier**: A text input field containing '1 - 16777215'.
- Subnets**: A section containing:
 - Network IPAM***: A dropdown menu with 'vCenter-ipam'.
 - CIDR***: A text input field containing '192.0.2.0/24'.
 - Allocation Pools**: A text input field containing 'xxx.xxx.xxx.xxx-xxx.xxx.xxx.xxx'.
 - Gateway***: A text input field containing '192.0.2.1'.
 - Service Address**: A text input field containing 'xxx.xxx.xxx.xxx'.
 - Auto Gateway**: A checked checkbox.
 - DHCP**: A checked checkbox.
 - DNS**: A checked checkbox.

At the bottom of the form, there are two buttons: 'Create' (blue) and 'Cancel' (light blue).

7. Click **Create** to create the virtual network.

The All Networks page is displayed as given in [Figure 3 on page 37](#).

Figure 3: All Networks Page



Delete Virtual Network

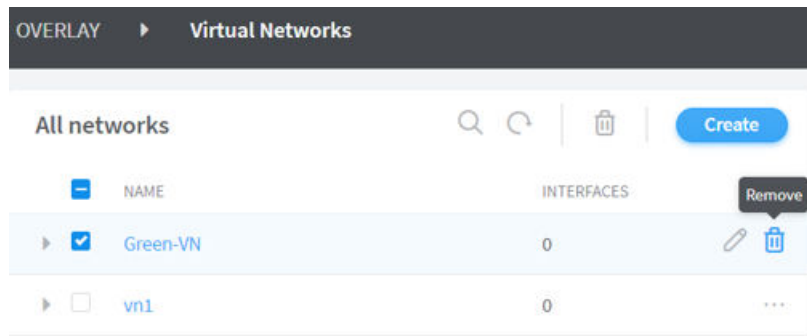
You can delete a virtual network from the Contrail Command UI.

Follow these steps to delete a virtual network.

1. Click **Overlay** > **Virtual Networks** to access the All Networks page.
2. Select the check box next to the virtual network you want to delete and move the mouse pointer to the end of the row.

Click the **Remove** icon to delete the virtual network. See [Figure 4 on page 38](#).

Figure 4: Delete a Virtual Network



NOTE: No virtual machines or templates must be using this virtual network before it is deleted.

A delete confirmation message is displayed.

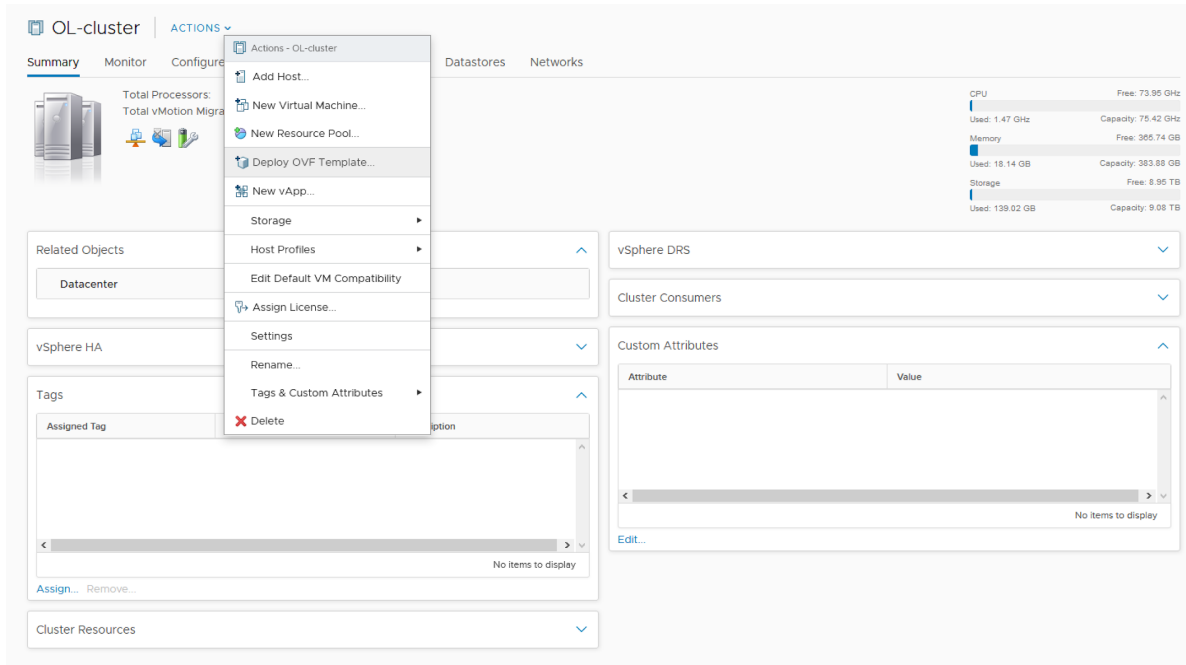
3. Click **Delete** to delete the selected network.

The virtual network is deleted.

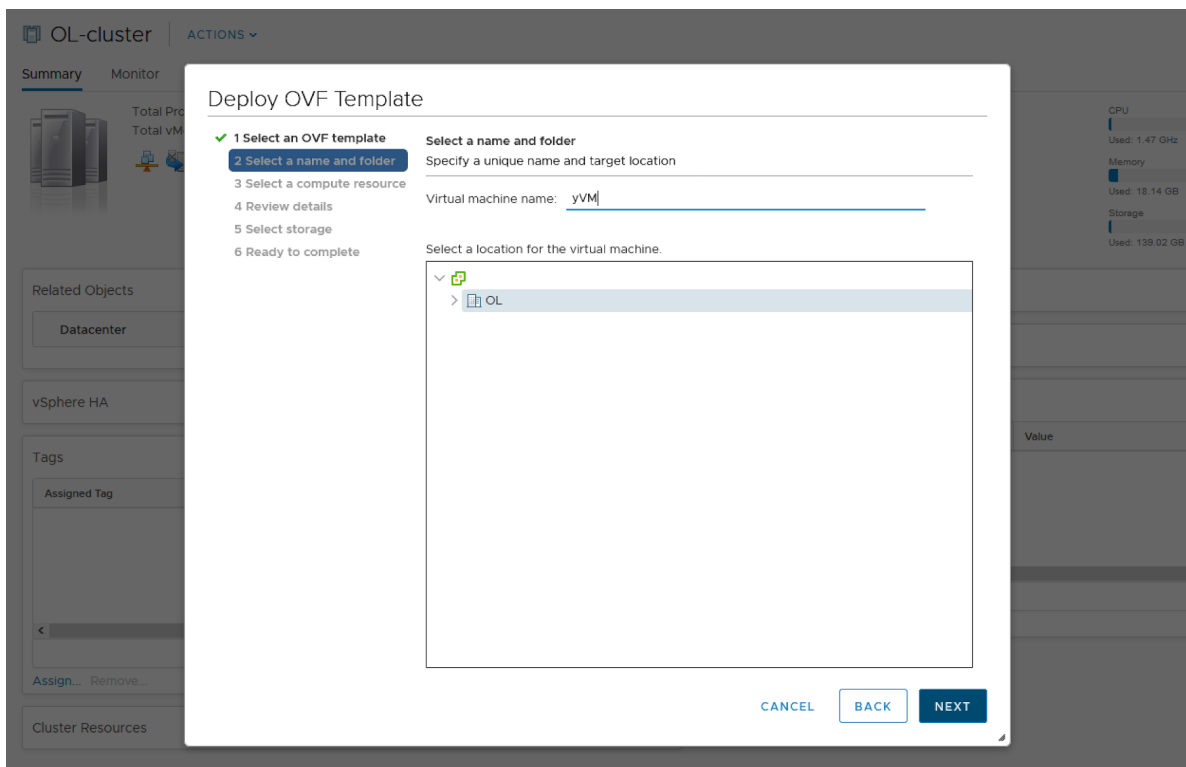
Create a Virtual Machine – vCenter UI

Use the vCenter client interface to create a virtual machine for your VMware vCenter Contrail integrated system. This section describes how to create a virtual machine using a virtual machine template from the vCenter client interface.

1. From the vCenter UI, right-click on the cluster and select **Deploy OVF Template**.

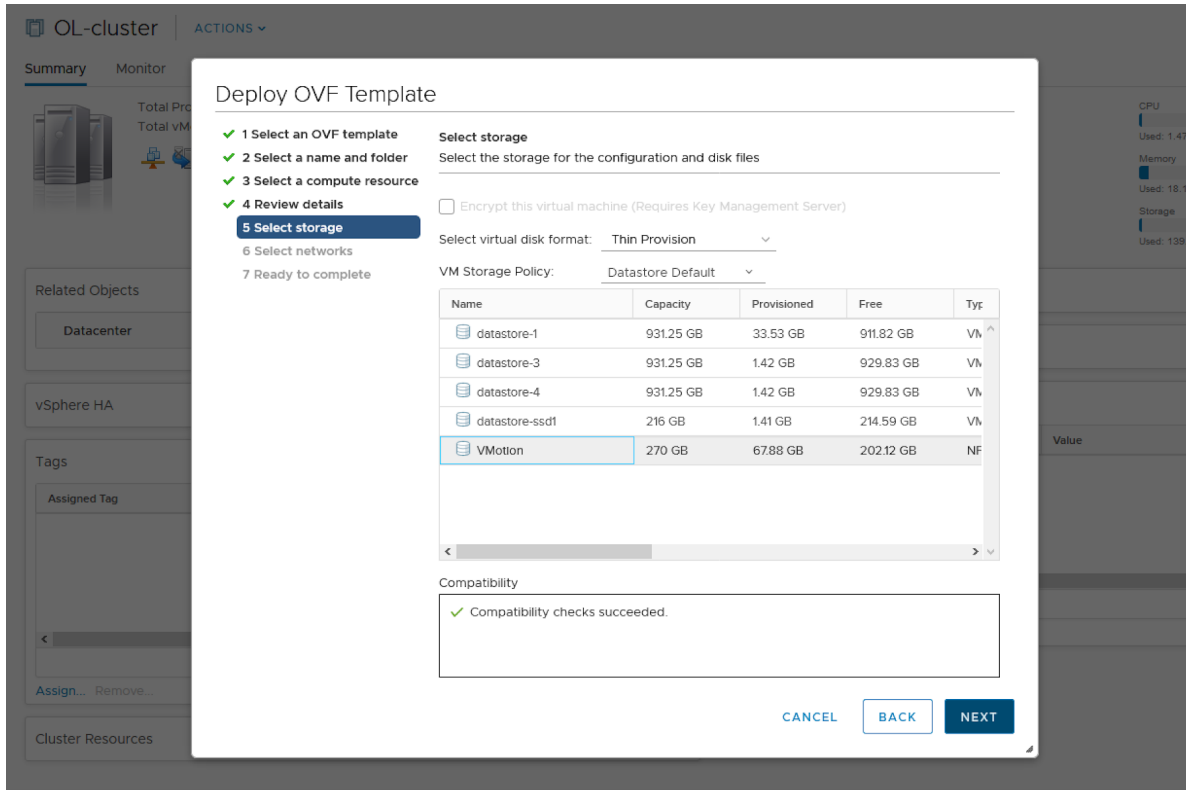


2. Select the virtual machine name and click **Next**.

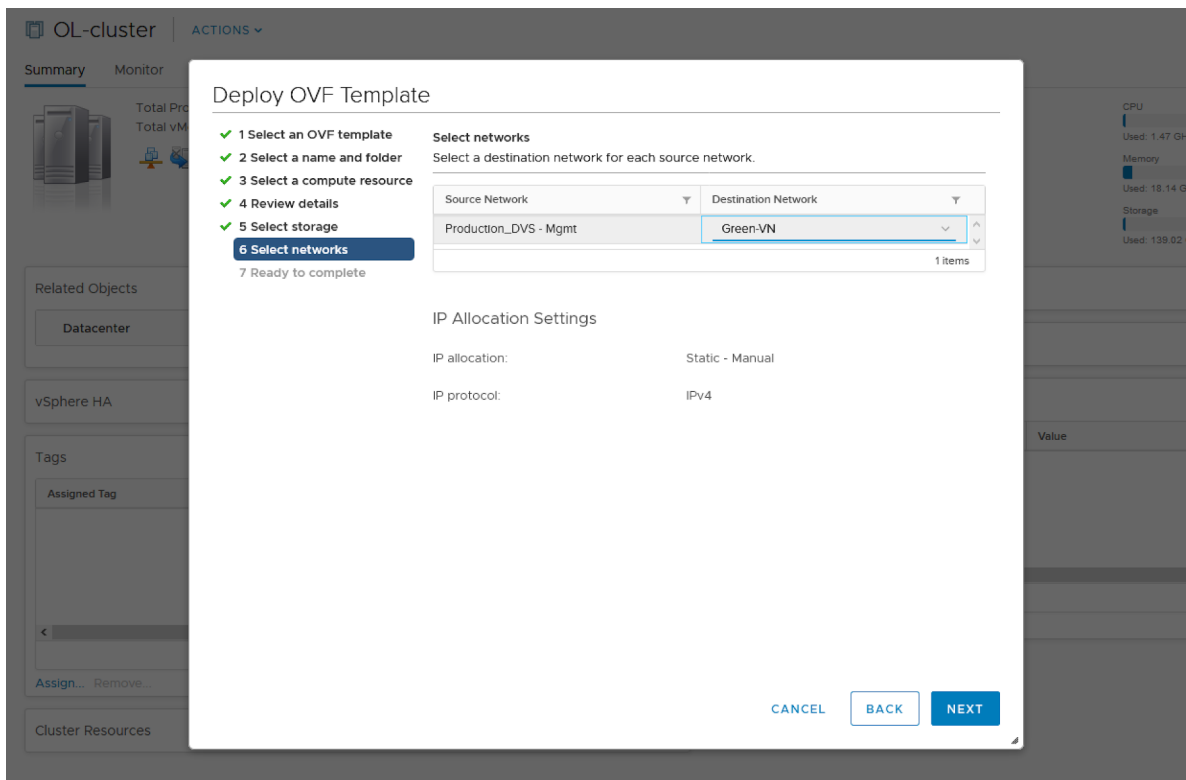


3. Select the compute resource on which you want to spawn the virtual machine and click **Next**.

4. Specify the storage location for the virtual machine and click **Next**.



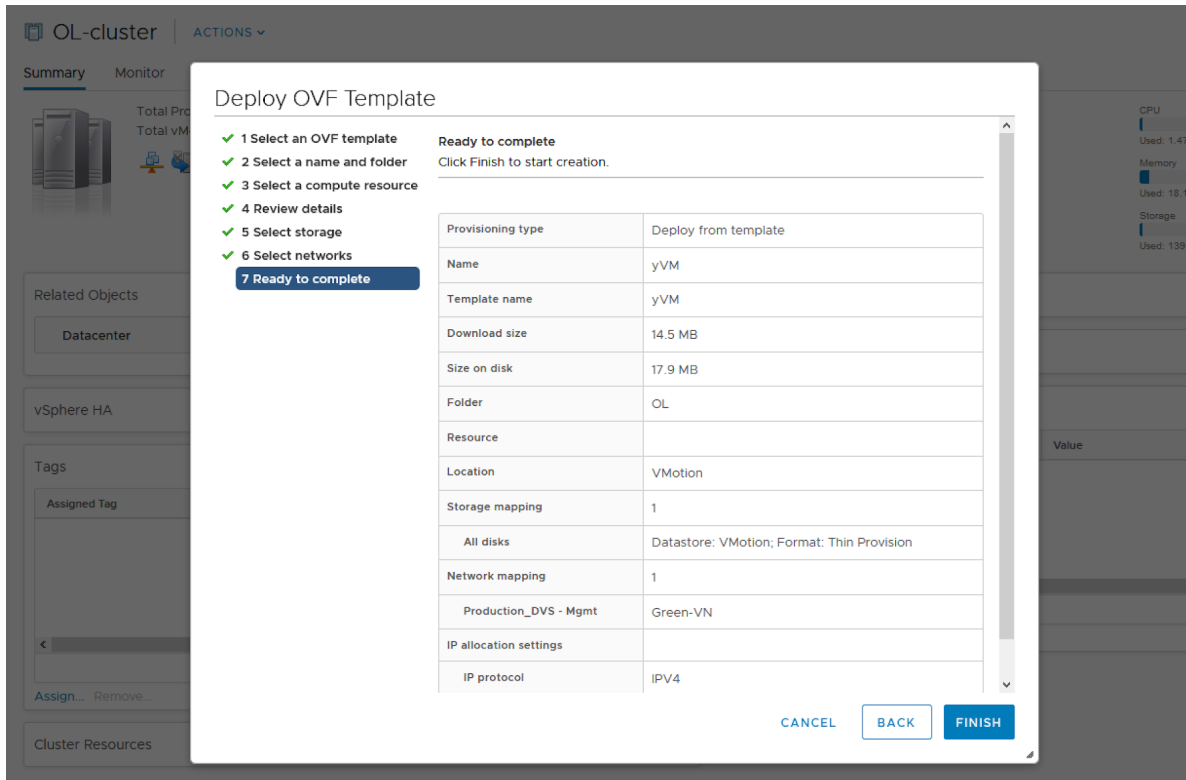
5. Select the **Destination Network**, Green-VN, that you created by using the Contrail Command UI.



6. On the **Ready to complete** window, review all of the virtual machine definitions that you have selected for the template.

If all the selections are correct, click **Finish**.

This spawns the virtual machine.



7. You return to the main vCenter UI window. Navigate to the console of the VM (yVM) you just spawned to see that it has been assigned an IP address from the Green-VN you created in the Contrail UI.


```

yVM
Enforce US Keyboard Layout View Fullscreen Send Ctrl+Alt+Delete

(=
//\ Core is distributed with ABSOLUTELY NO WARRANTY.
v_/_ www.tinycorelinux.com

tc@box:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr XX:XX:XX:XX:XX:XX
          inet addr:192.168.1.3  Bcast:XXX.XXX.X.XXX  Mask:XXX.XXX.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14  errors:0  dropped:0  overruns:0  frame:0
          TX packets:8  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:1000
          RX bytes:1374 (1.3 KiB)  TX bytes:1044 (1.0 KiB)

lo        Link encap:Local Loopback
          inet addr:XXX.0.0.1  Mask:XXX.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0  txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

tc@box:~$ _

```

Once the virtual machine is launched, you can view the virtual machine from the Contrail Command UI.

View Virtual Machine

You can view the virtual machines that you created by using the vCenter client, from the Contrail Command UI.

To view the virtual machines, click **Workloads > Instances**.

All the virtual machines are displayed in the Instances page.

RELATED DOCUMENTATION

| [Configuring Underlay Network for ContrailVM](#)

Integrating Contrail Release 5.0.X with VMware vRealize Orchestrator

IN THIS SECTION

● [Components of vRealize Orchestrator](#) | 43

A dedicated Contrail plugin is used to connect to VMware vRealize Orchestrator (vRO). Contrail Release 5.0 supported a Beta version of the plugin. Starting with Contrail Release 5.0.1, a fully supported version of the plugin is available. The plugin is used to view the Contrail controller configuration in the vRO inventory, and to modify configurations by using vRO workflows. You can also create network policies, security groups, and automate both simple and complex workflows by using vRO.

Components of vRealize Orchestrator

vRO consists of the following components:

- **vRO Inventory**—The vRO inventory displays the Contrail plugin and the Contrail node or endpoint. All Contrail plugin objects that represent your system are displayed in the vRO Inventory. Objects are displayed in a hierarchical order and are based on the Contrail schema.

With Release 5.0, Contrail inventory objects such as projects, security groups, virtual networks, network IPAMs, network policies, ports, floating IP pools, and service templates are displayed in the vRO inventory. Relevant API objects are also displayed in the vRO inventory.

- **vRO Workflows**—The vRO workflow is a process that manipulates objects in a vRO Inventory. Each plugin has a set of predefined workflows. vRO combines workflows from different plugins to create complex processes and manages them. Multiple workflows are used to create blueprints in vRA.

NOTE: VMware vCenter plugin workflows are represented as simple workflows in vRO plugin.

Contrail Workflows

You must connect to the Contrail controller or an endpoint before you create Contrail workflows. You must use **Create Contrail controller connection** to connect to an endpoint. You must use **Delete Contrail controller connection** to terminate a connection with an endpoint. Once you connect to the Control controller, the vRO plugin accesses the Contrail inventory objects and then updates the vRO inventory with the Contrail inventory objects.

The following workflows are defined for simple networking operations in Contrail Release 5.0:

- **Network**
 - Create network

- Delete network

NOTE: You must use the **Create network** workflow to create a network on the Contrail controller.

- Network Policy
 - Create network policy
 - Add rule to network policy
 - Remove network policy rule
 - Delete network policy
- Security policy
 - Create security group
 - Add rule to security group
 - Edit security group
 - Remove security group rule
 - Delete security group
- Service Instance
 - Add port tuple to service instance
 - Create service instance
 - Delete service instance
 - Remove port tuple from service instance
- Network IPAM
- Port
- Project
- Service Template
- Virtual Network
- Floating IP

- Create floating IP
- Delete floating IP
- Floating IP pools
 - Create floating IP pool
 - Delete floating IP pool
 - Edit floating IP pool

Starting with Contrail Release 5.0.1, the following workflows are also defined:

- Tag
 - Create global tag
 - Create tag in project
 - Delete tag
- Tag Type
 - Create tag type
 - Delete tag type
- Network Policy
 - Edit network policy rule
- Security policy
 - Edit security group rule
- Service Health Check
 - Create service health check
 - Add service instance to service health check
 - Remove service instance from service health check
 - Edit service health check
 - Delete service health check
- Project
 - Add application policy set to project

- Remove application policy set to project
- Add tag to project
- Remove tag from project
- Virtual Network
 - Add tag to virtual network
 - Remove tag from virtual network
- Virtual Machine Interface (VMI) - Port
 - Add tag to port
 - Remove tag from port
- Service Group
 - Create service group in policy management
 - Create service group in project
 - Add service to service group
 - Edit service of service group
 - Remove service from service group
 - Delete service group
- Address Group
 - Create global address group
 - Create address group in project
 - Add subnet to address group
 - Remove subnet from address group
 - Delete address group
 - Add label to address group
 - Remove label from address group
- Application Policy Set
 - Create global application policy set

- Create application policy set in project
- Add firewall policy to application policy set
- Remove firewall policy from application policy set
- Add tag to application policy set
- Remove tag from application policy set
- Delete application policy set
- Firewall Policy
 - Create global firewall policy
 - Create firewall policy in project
 - Add firewall rule to firewall policy
 - Remove firewall rule from firewall policy
 - Delete firewall policy
- Firewall Rule
 - Create global firewall rule
 - Create firewall rule in project
 - Edit firewall rule
 - Delete firewall rule

3

PART

Deploying Contrail Networking with OpenStack

[Configuring Virtual Networks | 49](#)

[Using Contrail Resources in Heat Templates | 64](#)

[QoS Support in Contrail Networking | 69](#)

[Load Balancers | 80](#)

[Optimizing Contrail Networking | 112](#)

[Contrail Networking OpenStack Analytics | 114](#)

[Contrail OpenStack APIs | 122](#)

Configuring Virtual Networks

IN THIS CHAPTER

- Creating Projects in OpenStack for Configuring Tenants in Contrail | 49
- Creating a Virtual Network with OpenStack Contrail | 51
- Creating an Image for a Project in OpenStack Contrail | 55
- Using Security Groups with Virtual Machines (Instances) | 58

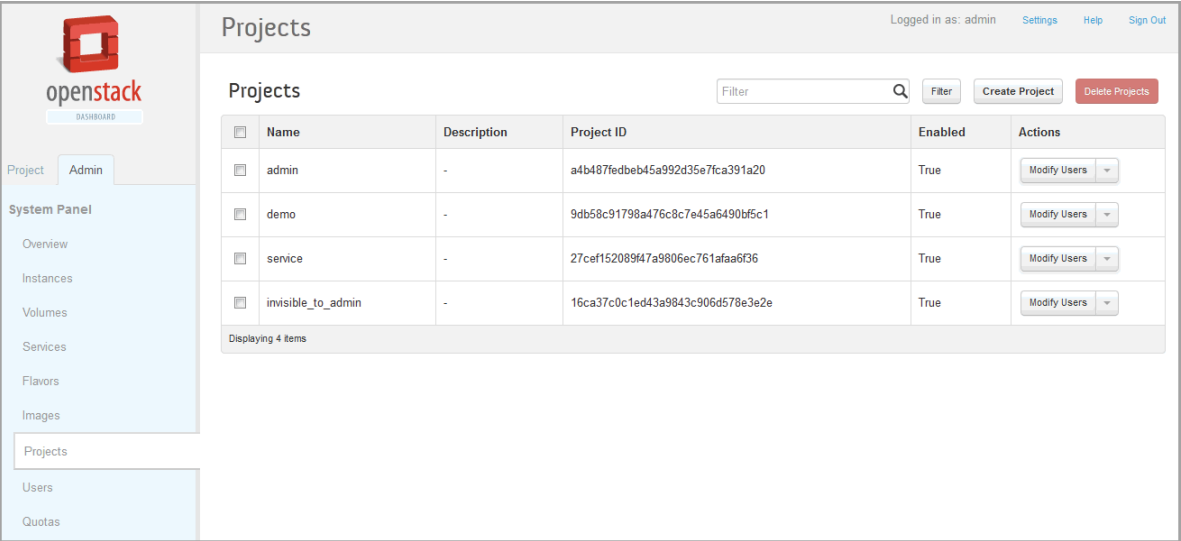
Creating Projects in OpenStack for Configuring Tenants in Contrail

In Contrail, a tenant configuration is called a project. A project is created for each set of virtual machines (VMs) and virtual networks (VNs) that are configured as a discrete entity for the tenant.

Projects are created, managed, and edited at the OpenStack **Projects** page.

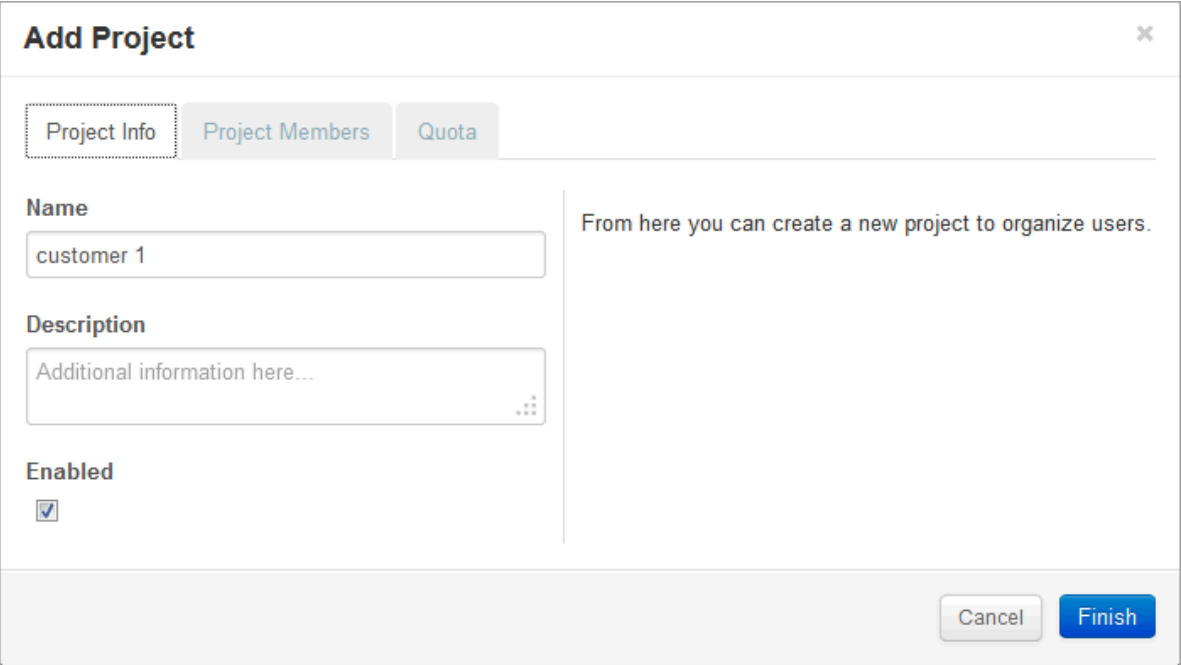
1. Click the **Admin** tab on the OpenStack dashboard, then click the **Projects** link to access the **Projects** page; see [Figure 5 on page 50](#).

Figure 5: OpenStack Projects



2. In the upper right, click the **Create Project** button to access the **Add Project** window; see [Figure 6 on page 50](#).

Figure 6: Add Project



3. In the **Add Project** window, on the **Project Info** tab, enter a **Name** and a **Description** for the new project, and select the **Enabled** check box to activate this project.

4. In the **Add Project** window, select the **Project Members** tab, and assign users to this project. Designate each user as **admin** or as **Member**.
As a general rule, one person should be a super user in the **admin** role for all projects and a user with a **Member** role should be used for general configuration purposes.
5. Click **Finish** to create the project.

Refer to OpenStack documentation for more information about creating and managing projects.

RELATED DOCUMENTATION

Creating a Virtual Network with Juniper Networks Contrail

[Creating a Virtual Network with OpenStack Contrail | 51](#)

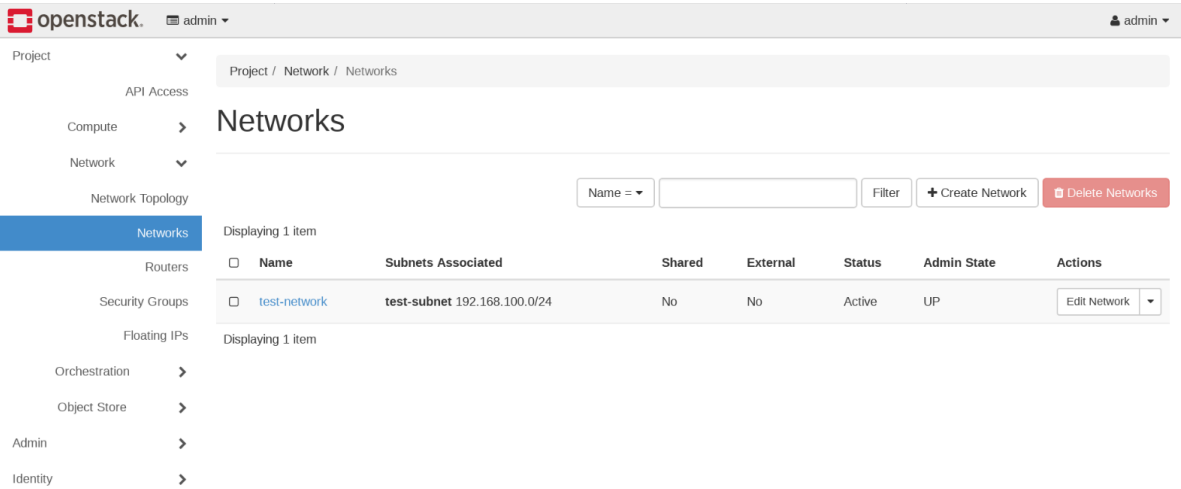
[OpenStack documentation](#)

Creating a Virtual Network with OpenStack Contrail

You can create virtual networks in Contrail Networking from the OpenStack. The following procedure shows how to create a virtual network when using OpenStack.

1. To create a virtual network when using OpenStack Contrail, select **Project > Network > Networks**. The **Networks** page is displayed. See [Figure 7 on page 52](#).

Figure 7: Networks Page



2. Click **Create Network**. The **Create Network** window is displayed. See [Figure 8 on page 52](#) and [Figure 9 on page 53](#).

Figure 8: Create Networks

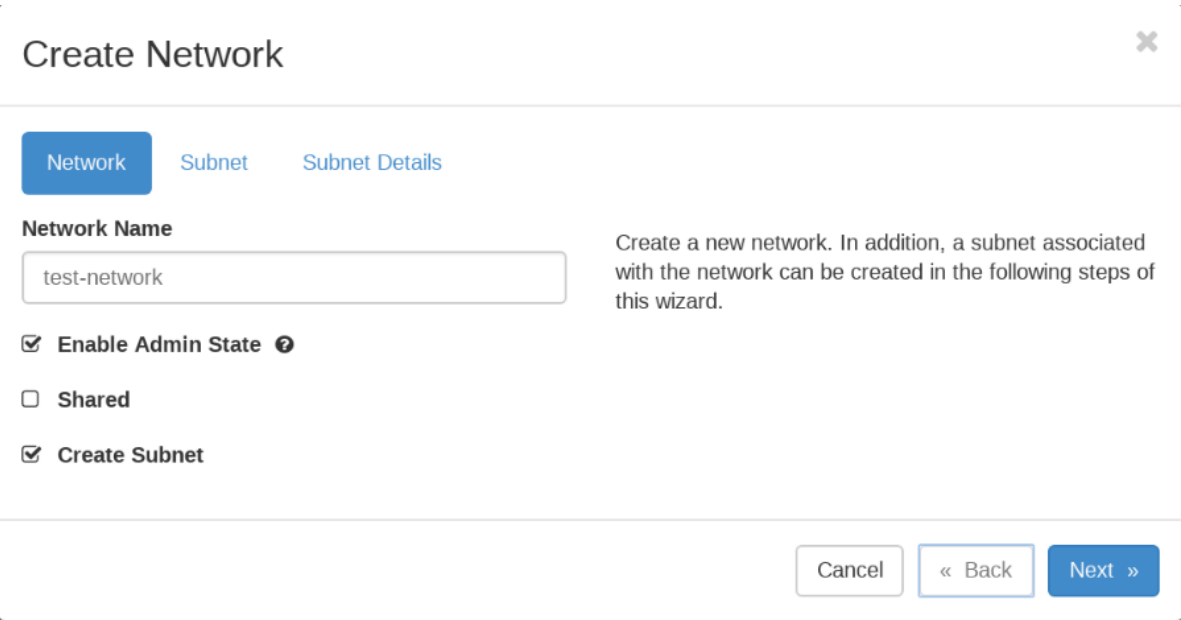


Figure 9: Subnet and Gateway Details

Create Network

Network

Subnet

Subnet Details

Subnet Name

Network Address ?

IP Version

IPv4

Gateway IP ?

☐ Disable Gateway

Creates a subnet associated with the network. You need to enter a valid "Network Address" and "Gateway IP". If you did not enter the "Gateway IP", the first value of a network will be assigned by default. If you do not want gateway please check the "Disable Gateway" checkbox. Advanced configuration is available by clicking on the "Subnet Details" tab.

Cancel

« Back

Next »

3. Click the **Network** and **Subnet** tabs to complete the fields in the **Create Network** window. See field descriptions in [Table 4 on page 53](#).

Table 4: Create Network Fields

Field	Description
Network Name	Enter a name for the network.
Subnet Name	Enter a name for the subnetwork.
Network Address	Enter the network address in CIDR format.
IP Version*	Select IPv4 or IPv6.

Table 4: Create Network Fields *(Continued)*

Field	Description
Gateway IP	Optionally, enter an explicit gateway IP address for the IP address block. Check the Disable Gateway box if no gateway is to be used.

4. Click the **Subnet Details** tab to specify the Allocation Pool, DNS Name Servers, and Host Routes.

Figure 10: Additional Subnet Attributes

Create Network

Network

Subnet

Subnet Details

☒ Enable DHCP

Specify additional attributes for the subnet.

Allocation Pools ?

DNS Name Servers ?

Host Routes ?

Cancel

« Back

Create

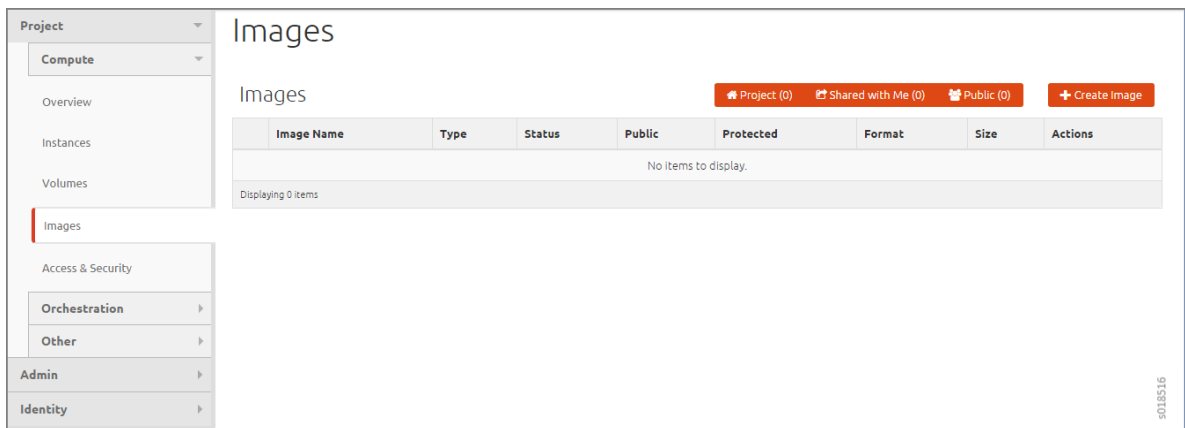
5. To save your network, click **Create** , or click **Cancel** to discard your work and start over.

Creating an Image for a Project in OpenStack Contrail

To specify an image to upload to the Image Service for a project in your system by using the OpenStack dashboard:

1. In OpenStack, select **Project > Compute > Images**. The Images window is displayed. See [Figure 11 on page 55](#).

Figure 11: OpenStack Images Window



2. Make sure you have selected the correct project to which you are associating an image.
3. Click **Create Image**.

The **Create An Image** window is displayed. See [Figure 12 on page 56](#).

Figure 12: OpenStack Create An Image Window

Create An Image

Name *

Description

Image Source

Image Location ▼

Image Location ?

http://example.com/image.iso

Format *

Select format ▼

Architecture

Minimum Disk (GB) ?

Minimum RAM (MB) ?

☐ Public

☐ Protected

Description:

Currently only images available via an HTTP URL are supported. The image location must be accessible to the Image Service. Compressed image binaries are supported (.zip and .tar.gz.)

Please note: The Image Location field MUST be a valid and direct URL to the image binary. URLs that redirect or serve error pages will result in unusable images.

s018515

Cancel

Create Image

4. Complete the fields to specify your image. [Table 5 on page 57](#) describes each of the fields on the window.

NOTE: Only images available through an HTTP URL are supported, and the image location must be accessible to the Image Service. Compressed image binaries are supported (*.zip and *.tar.gz).

Table 5: Create an Image Fields

Field	Description
Name	Enter a name for this image.
Description	Enter a description for the image.
Image Source	<p>Select Image File or Image Location.</p> <p>If you select Image File, you are prompted to browse to the local location of the file.</p>
Image Location	Enter an external HTTP URL from which to load the image. The URL must be a valid and direct URL to the image binary. URLs that redirect or serve error pages result in unusable images.
Format	<p>Required field. Select the format of the image from a list:</p> <ul style="list-style-type: none"> AKI- Amazon Kernel Image AMI- Amazon Machine Image ARI- Amazon Ramdisk Image ISO- Optical Disk Image QCOW2- QEMU Emulator Raw- An unstructured image format VDI- Virtual Disk Image VHD- Virtual Hard Disk VMDK- Virtual Machine Disk
Architecture	Enter the architecture.

Table 5: Create an Image Fields *(Continued)*

Field	Description
Minimum Disk (GB)	Enter the minimum disk size required to boot the image. If you do not specify a size, the default is 0 (no minimum).
Minimum Ram (MB)	Enter the minimum RAM required to boot the image. If you do not specify a size, the default is 0 (no minimum).
Public	Select this check box if this is a public image. Leave unselected for a private image.
Protected	Select this check box for a protected image.

5. When you are finished, click **Create Image**.

Using Security Groups with Virtual Machines (Instances)

IN THIS SECTION

- [Security Groups Overview | 58](#)
- [Creating Security Groups and Adding Rules | 59](#)

Security Groups Overview

A **security group** is a container for security group rules. Security groups and security group rules allow administrators to specify the type of traffic that is allowed to pass through a port. When a virtual machine (VM) is created in a virtual network (VN), a security group can be associated with the VM when it is launched. If a security group is not specified, a port is associated with a default security group. The default security group allows both ingress and egress traffic. Security rules can be added to the default security group to change the traffic behavior.

Creating Security Groups and Adding Rules

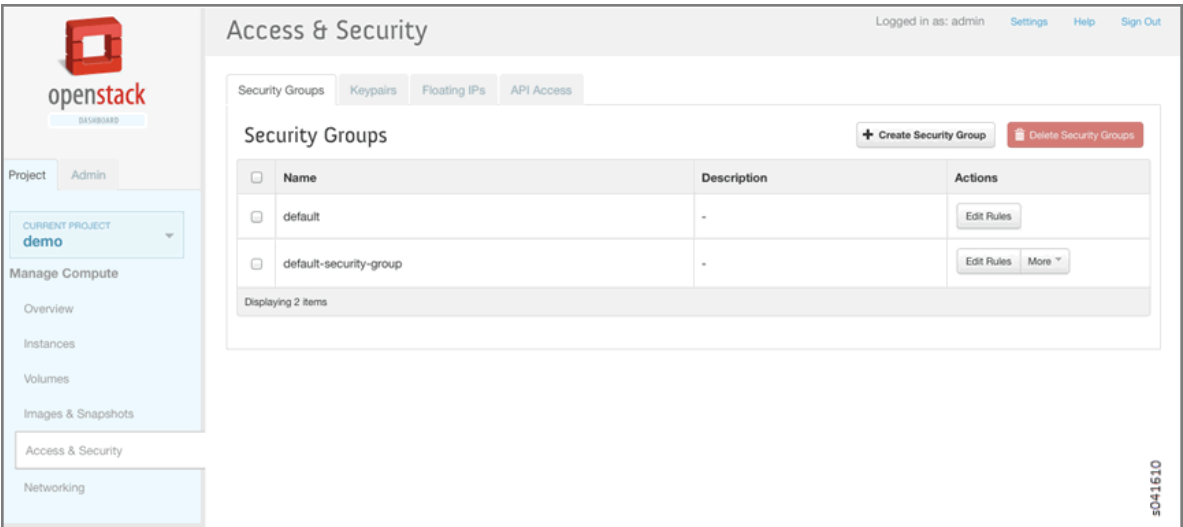
A default security group is created for each project. You can add security rules to the default security group and you can create additional security groups and add rules to them. The security groups are then associated with a VM, when the VM is launched or at a later date.

To add rules to a security group:

- 1. From the OpenStack interface, click the **Project** tab, select **Access & Security**, and click the **Security Groups** tab.

Any existing security groups are listed under the **Security Groups** tab, including the default security group; see [Figure 13 on page 59](#).

Figure 13: Security Groups



- 2. Select the **default-security-group** and click **Edit Rules** in the **Actions** column.
- The **Edit Security Group Rules** window is displayed; see [Figure 14 on page 60](#). Any rules already associated with the security group are listed.

Figure 14: Edit Security Group Rules

The screenshot displays the OpenStack dashboard interface for editing security group rules. The top navigation bar includes the OpenStack logo, a 'Dashboard' button, and user information (Logged in as: admin) with links for Settings, Help, and Sign Out. The main content area is titled 'Edit Security Group Rules' and features a table of existing rules. The table has columns for IP Protocol, From Port, To Port, Source, and Actions. Two rules are listed: a TCP rule on port 22 and an ICMP rule on port 0, both with a source of 8.8.8.0/24 (CIDR). Each rule has a 'Delete Rule' button in the Actions column. Above the table are buttons for '+ Add Rule' and 'Delete Rules'. The left sidebar contains navigation links for Project, Admin, and various management categories like Compute, Storage, and Networking.

<input type="checkbox"/>	IP Protocol	From Port	To Port	Source	Actions
<input type="checkbox"/>	TCP	22	22	8.8.8.0/24 (CIDR)	Delete Rule
<input type="checkbox"/>	ICMP	0	65535	8.8.8.0/24 (CIDR)	Delete Rule

Displaying 2 items

3. Click **Add Rule** to add a new rule; see [Figure 15 on page 61](#).

Figure 15: Add Rule

Add Rule

IP Protocol

ICMP

Type

0

Code

0

Source

CIDR

CIDR

Security Group

0.0.0.0/0

Description:

Rules define which traffic is allowed to instances assigned to the security group. A security group rule consists of three main parts:

Protocol:

You must specify the desired IP protocol to which this rule will apply; the options are TCP, UDP, or ICMP.

Open Port/Port Range:

For TCP and UDP rules you may choose to open either a single port or a range of ports. Selecting the "Port Range" option will provide you with space to provide both the starting and ending ports for the range. For ICMP rules you instead specify an ICMP type and code in the spaces provided.

Source:

You must specify the source of the traffic to be allowed via this rule. You may do so either in the form of an IP address block (CIDR) or via a source group (Security Group). Selecting a security group as the source will allow any other instance in that security group access to any other instance via this rule.

Cancel

Add

Table 6: Add Rule Fields

Column	Description
IP Protocol	Select the IP protocol to apply for this rule: TCP, UDP, ICMP.
From Port	Select the port from which traffic originates to apply this rule. For TCP and UDP, enter a single port or a range of ports. For ICMP rules, enter an ICMP type code.
To Port	The port to which traffic is destined that applies to this rule, using the same options as in the From Port field.

Table 6: Add Rule Fields (Continued)

Column	Description
Source	Select the source of traffic to be allowed by this rule. Specify subnet—the CIDR IP address or address block of the inter-domain source of the traffic that applies to this rule, or you can choose security group as source. Selecting security group as source allows any other instance in that security group access to any other instance via this rule.

4. Click **Create Security Group** to create additional security groups.

The **Create Security Group** window is displayed; see [Figure 16 on page 62](#).

Each new security group has a unique 32-bit security group ID and an ACL is associated with the configured rules.

Figure 16: Create Security Group

5. When an instance is launched, there is an opportunity to associate a security group; see [Figure 17 on page 63](#).

In the **Security Groups** list, select the security group name to associate with the instance.

Figure 17: Associate Security Group at Launch Instance

Launch Instance [X]

Details | Access & Security | Networking | Volume Options | Post-Creation

Keypair

No keypairs available. [v] +

Security Groups

☒ SG1

☐ default

☐ default-security-group

Control access to your instance via keypairs, security groups, and other mechanisms.

Cancel Launch

6. You can verify that security groups are attached by viewing the `SgListReq` and `IntfReq` associated with the `agent.xml`.

Using Contrail Resources in Heat Templates

IN THIS CHAPTER

- [Using the Contrail Heat Template | 64](#)

Using the Contrail Heat Template

IN THIS SECTION

- [Introduction to Heat | 64](#)
- [Heat Architecture | 65](#)
- [Support for Heat Version 2 Resources | 65](#)
- [Heat Version 2 with Service Templates and Port Tuple Sample Workflow | 66](#)
- [Example: Creating a Service Template Using Heat | 66](#)

Heat is the orchestration engine of the OpenStack program. Heat enables launching multiple cloud applications based on templates that are comprised of text files.

Introduction to Heat

A Heat template describes the infrastructure for a cloud application, such as networks, servers, floating IP addresses, and the like, and can be used to manage the entire life cycle of that application.

When the application infrastructure changes, the Heat templates can be modified to automatically reflect those changes. Heat can also delete all application resources if the system is finished with an application.

Heat templates can record the relationships between resources, for example, which networks are connected by means of policy enforcements, and consequently call OpenStack REST APIs that create

the necessary infrastructure, in the correct order, needed to launch the application managed by the Heat template.

Heat Architecture

Heat is implemented by means of Python applications, including the following:

- `heat-client`—The CLI tool that communicates with the `heat-api` application to run Heat APIs.
- `heat-api`—Provides an OpenStack native REST API that processes API requests by sending them to the Heat engine over remote procedure calls (RPCs).
- `heat-engine`—Responsible for orchestrating the launch of templates and providing events back to the API consumer.

Support for Heat Version 2 Resources

Starting with Contrail Release 3.0.2, Contrail Heat resources and templates are autogenerated from the Contrail schema, using Heat Version 2 resources. Contrail Release 3.0.2 is the minimum required version for using Heat with Contrail in 3.x releases. The Contrail Heat Version 2 resources are of the following hierarchy: `OS::ContrailV2::<ResourceName>`.

The generated resources and templates are part of the Contrail Python package, and are located in the following directory in the target installation:

`/usr/lib/python2.7/dist-packages/vnc_api/gen/heat/`

The `heat/` directory has the following subdirectories:

- **resources/**—Contains all the resources for the contrail-heat plugin, which runs in the context of the Heat engine service.
- **templates/**—Contains sample templates for each resource. Each sample template presents every possible parameter in the schema. Use the sample templates as a reference when you build up more complex templates for your network design.
- **env/**—Contains the environment for input to each template.

The following contains a list of all the generated plug-in resources that are supported by contrail-heat :

https://github.com/tungstenfabric/tf-heat-plugin/tree/master/contrail_heat/new_templates

Deprecation of Heat Version 1 Resources

Heat Version 1 resources within the hierarchy `OS::Contrail::<ResourceName>` are being deprecated, and you should not create new service templates using the Heat Version 1 templates.

Heat Version 2 with Service Templates and Port Tuple Sample Workflow

With Contrail service templates Version 2, the user can create ports and bind them to a virtual machine (VM)-based service instance, by means of a port-tuple object. All objects created with the Version 2 service template are directly visible to the Contrail Heat engine, and are directly managed by Heat.

The following shows the basic workflow steps for creating a port tuple and service instance that will be managed by Heat:

1. Create a service template. Select 2 in the Version field.
2. Create a service instance for the service template just created.
3. Create a port-tuple object.
4. Create ports, using Nova VM launch or without a VM launch.
5. Label each port as left, right, mgmt, and so on, and add the ports to the port-tuple object.

Use a unique label for each of the ports in a single port tuple. The labels named left and right are used for forwarding.

6. Link the port tuple to a service instance.
7. Launch the service instance.

Example: Creating a Service Template Using Heat

The following is an example of how to create a service template using Heat.

1. Define a template to create the service template.

```
service_template.yaml
heat_template_version: 2013-05-23
description: >
  HOT template to create a service template
parameters:
  name:
    type: string
    description: Name of service template
  mode:
    type: string
    description: service mode
  type:
    type: string
```

```

        description: service type
    image:
        type: string
        description: Name of the image
    flavor:
        type: string
        description: Flavor
    service_interface_type_list:
        type: string
        description: List of interface types
    shared_ip_list:
        type: string
        description: List of shared ip enabled-â€œdisabled
    static_routes_list:
        type: string
        description: List of static routes enabled-â€œdisabled

resources:
    service_template:
        type: OS::ContrailV2::ServiceTemplate
        properties:
            name: { get_param: name }
            service_mode: { get_param: mode }
            service_type: { get_param: type }
            image_name: { get_param: image }
            flavor: { get_param: flavor }
            service_interface_type_list: { "Fn::Split" : [ ",", Ref:
service_interface_type_list ] }
            shared_ip_list: { "Fn::Split" : [ ",", Ref: shared_ip_list ] }
            static_routes_list: { "Fn::Split" : [ ",", Ref: static_routes_list ] }
        outputs:
            service_template_fq_name:
                description: FQ name of the service template
                value: { get_attr: [ service_template, fq_name] }

}

```

2. Create an environment file to define the values to put in the variables in the template file.

```
service_template.env
```

```
parameters:
```

```

name: contrail_svc_temp

mode: transparent

type: firewall

image: cirros

flavor: m1.tiny

service_interface_type_list: management,left,right,other

shared_ip_list: True,True,False,False

static_routes_list: False,True,False,False

```

3. Create the Heat stack by launching the template and the environment file, using the following command:

```
heat stack create stack1 -f service_template.yaml -e service_template.env
```

OR use this command for recent versions of OpenStack

```
openstack stack create -e <env-file-name> -t <template-file-name> <stack-name>
```

RELATED DOCUMENTATION

| *Service Chain Version 2 with Port Tuple*

CHAPTER 5

QoS Support in Contrail Networking

IN THIS CHAPTER

- [Quality of Service in Contrail | 69](#)
- [Configuring Network QoS Parameters | 77](#)

Quality of Service in Contrail

IN THIS SECTION

- [Overview: Quality of Service | 69](#)
- [Contrail QoS Model | 70](#)
- [Features of Fabric Interfaces | 70](#)
- [QoS Configuration Parameters for Provisioning | 70](#)
- [Configuring QoS in Contrail Networking Release 5.0 and Later | 70](#)
- [Queuing Implementation | 71](#)
- [Contrail QoS Configuration Objects | 72](#)
- [Example: Mapping Traffic to Forwarding Classes | 73](#)
- [QoS Configuration Object Marking on the Packet | 74](#)
- [Queuing | 75](#)

Overview: Quality of Service

Quality of service (QoS) in networking provides the ability to control reliability, bandwidth, latency, and other traffic management features. Network traffic can be marked with QoS bits (DSCP, 802.1p, and MPLS EXP) that intermediate network switches and routers can use to provide service guarantees.

Contrail QoS Model

The QoS model in Contrail Networking has the following features:

- All packet forwarding devices, such as vRouter and the gateway, combine to form a system.
- Interfaces to the system are the ports from which the system sends and receives packets, such as tap interfaces and physical ports.
- Fabric interfaces are where the overlay traffic is tunneled.
- QoS is applied at the ingress to the system, for example, upon traffic from the interfaces to the fabric.
- At egress, packets are stripped of their tunnel headers and sent to interface queues, based on the forwarding class. No marking from the outer packet to the inner packet is considered at this time.

Features of Fabric Interfaces

Fabric interfaces, unlike other interfaces, are always shared. Therefore, fabric interfaces are common property. Consequently, traffic classes and QoS marking on the fabric must be controlled by the system administrator. The administrator might choose to provision different classes of service on the fabric.

In Contrail, classes of service are determined by both of the following:

- Queueing on the fabric interface, including queues, scheduling of queues, and drop policies, and
- forwarding class, a method of marking that controls how packets are sent to the fabric, including marking and identifying which queue to use.

Tenants can define which forwarding class their traffic can use, deciding which packets use which forwarding class. The Contrail QoS configuration object has a mapping table, mapping the incoming DSCP or 802.1p value to the forwarding class mapping.

The QoS configuration can also be applied to a virtual network, an interface, or a network policy.

QoS Configuration Parameters for Provisioning

Configuring QoS in Contrail Networking Release 5.0 and Later

This section describes how to provision QoS in Contrail Networking release 5.0 and later.

1. Define the hardware queues and priority group in the **instances.yaml** file under the vrouter role as shown below.

```
nodeh5:
  ip: 10.xxx.xxx.109
  provider: bms
  roles:
    vrouter:
      VRUTER_GATEWAY: 192.168.1.45
      PRIORITY_ID: 0,1,2,3,4,5,6,7
      PRIORITY_BANDWIDTH: 0,10,0,20,0,30,0,40
      PRIORITY_SCHEDULING: strict,rr,strict,rr,strict,rr,strict,rr
      QOS_QUEUE_ID: 3,11,18,28,36,43,61,53
      QOS_LOGICAL_QUEUES: "[ 1, 6-10, 12-15];[40-46];[70-74, 75, 80-95];[115];[140-143,
145];[175];[245];[215]"
      QOS_DEF_HW_QUEUE: True
      openstack_compute:
```

2. In the already provisioned setup, define the QoS configuration in the **/etc/contrail/common_vrouter.env** file as shown in the following sample.

```
PRIORITY_ID=0,1,2,3,4,5,6,7
PRIORITY_BANDWIDTH=0,10,0,20,0,30,0,40
PRIORITY_SCHEDULING=strict,rr,strict,rr,strict,rr,strict,rr
QOS_QUEUE_ID=3,11,18,28,36,43,61,53
QOS_LOGICAL_QUEUES="[ 1, 6-10, 12-15];[40-46];[70-74, 75, 80-95];[115];[140-143, 145];[175];
[245];[215]"
QOS_DEF_HW_QUEUE=True
```

3. Execute the execute `docker-compose up -d` under `/etc/contrail/vrouter/` command.

Queuing Implementation

The vRouter provides the infrastructure to use queues supplied by the network interface, a method that is also called hardware queueing. Network interface cards (NICs) that implement hardware queueing have their own set of scheduling algorithms associated with the queues. The Contrail implementation is designed to work with most NICs, however, the method is tested only on an Intel-based 10G NIC, also called Niantic.

Contrail QoS Configuration Objects

Contrail QoS configuration objects include the:

- forwarding class
- QoS configuration object (qos-config)

The forwarding class object specifies parameters for marking and queuing, including:

- The DSCP, 802.1p, and MPLS EXP values to be written on packets.
- The queue index to be used for the packet.

The QoS configuration object specifies a mapping from DSCP, 802.1p, and MPLS EXP values to the corresponding forwarding class.

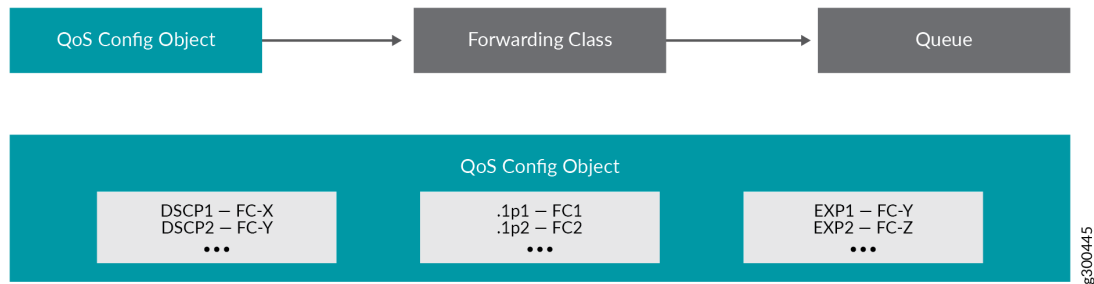
The QoS configuration has an option to specify the default forwarding class ID to use to select the forwarding class for all unspecified DSCP, 802.1p, and MPLS EXP values.

If the default forwarding class ID is not specified by the user, it defaults to the forwarding class with ID 0.

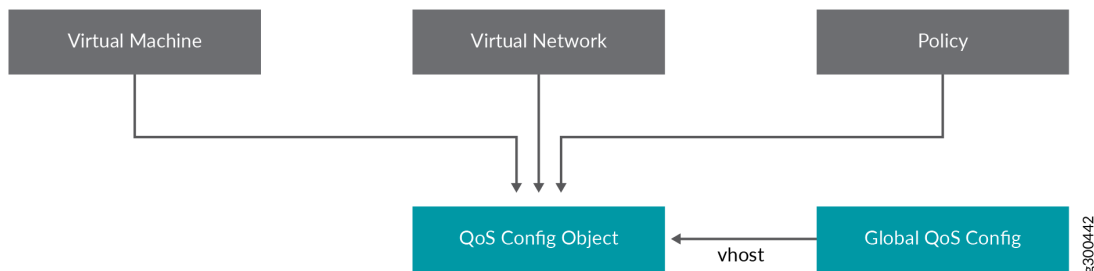
Processing of QoS marked packets to look up the corresponding forwarding class to be applied works as follows:

- For an IP packet, the DSCP map is used .
- For a Layer 2 packet, the 802.1p map is used.
- For an MPLS-tunneled packet with MPLS EXP values specified, the EXP bit value is used with the MPLS EXP map.
- If the QoS configuration is untrusted, only the default forwarding class is specified, and all incoming values of the DSCP, 802.1p, and EXP bits in the packet are mapped to the same default forwarding class.

[Figure 18 on page 73](#) shows the processing of QoS packets.

Figure 18: Processing of QoS Packets

A virtual machine interface, virtual network, and network policy can refer to the QoS configuration object. The QoS configuration object can be specified on the vhost so that underlay traffic can also be subjected to marking and queuing. See [Figure 19 on page 73](#).

Figure 19: Referring to the QoS Object

Example: Mapping Traffic to Forwarding Classes

This example shows how traffic forwarding classes are defined and how the QoS configuration object is defined to map the QoS bits to forwarding classes.

[Table 7 on page 73](#) shows two forwarding class objects defined. FC1 marks the traffic with high priority values and queues it to Queue 0. FC2 marks the traffic as best effort and queues the traffic to Queue 1.

Table 7: Forwarding Class Mapping

Name	ID	DSCP	802.1p	MPLS EXP	Queue
FC1	1	10	7	7	0

Table 7: Forwarding Class Mapping (Continued)

Name	ID	DSCP	802.1p	MPLS EXP	Queue
FC2	2	38	0	0	1

In [Table 8 on page 74](#), the QoS configuration object DSCP values of 10, 18, and 26 are mapped to a forwarding class with ID 1, which is forwarding class FC1. All other IP packets are mapped to the forwarding class with ID 2, which is FC2. All traffic with an 802.1p value of 6 or 7 are mapped to forwarding class FC1, and the remaining traffic is mapped to FC2.

Table 8: QoS Configuration Object Mapping

DSCP	Forwarding Class ID	802.1p	Forwarding Class ID	MPLS EXP	Forwarding Class ID
10	1	6	1	5	1
18	1	7	1	7	1
26	1	*	2	*	1
*	2				

QoS Configuration Object Marking on the Packet

The following sections describes how QoS configuration object marking is handled in various circumstances.

Traffic Originated by a Virtual Machine Interface

- If a VM interface sends an IP packet to another VM in a remote compute node, the DSCP value in the IP header is used to look into the qos-config table, and the tunnel header is marked with DSCP, 802.1p, and MPLS EXP bits as specified by the forwarding class.
- If a VM sends a Layer 2 non-IP packet with an 802.1p value, the 802.1p value is used to look into the qos-config table, and the corresponding forwarding class DSCP, 802.1p, and MPLS EXP value is written to the tunnel header.

- If a VM sends an IP packet to a VM in same compute node, the packet headers are not changed while forwarding. The original packet remains unchanged.

Traffic Destined to a Virtual Machine Interface

For traffic destined to a VMI, if a tunneled packet is received, the tunnel headers are stripped off and the packet is sent to the interface. No marking is done from the outer packet to inner packet.

Traffic from a vhost Interface

The QoS configuration can be applied on IP traffic coming from a vhost interface. The DSCP value in the packet is used to look into the qos-config object specified on the vhost, and the corresponding forwarding class DSCP and 802.1p values are overwritten on the packet.

Traffic from fabric interface

The QoS configuration can be applied while receiving the packet on an Ethernet interface of a compute node, and the corresponding forwarding class DSCP and 802.1p values are overwritten on the packet.

QoS Configuration Priority by Level

The QoS configuration can be specified at different levels.

The levels that can be configured with QoS and their order of priority:

1. in policy
2. on virtual-network
3. on virtual-machine-interface

Queuing

Contrail Networking supports QoS. These sections provide an overview of the queuing features available in Contrail Networking.

The queue to which a packet is sent is specified by the forwarding class.

Queue Selection in Datapath

In vRouter, in the data path, the forwarding class number specifies the actual physical hardware queue to which the packet needs to be sent, not to a logical selection as in other parts of Contrail. There is a

mapping table in the vRouter configuration file, to translate the physical queue number from the logical queue number.

Hardware Queueing in Linux kernel based vRouter

If Xmit-Packet-Steering (XPS) is enabled, the kernel chooses the queue, from those available in a list of queues. If the kernel selects the queue, packets will not be sent to the vRouter-specified queue.

To disable this mapping:

- have a kernel without CONFIG_XPS option
- write zeros to the mapping file in `/sys/class/net//queues/tx-X/xps_cpus`.

When this mapping is disabled, the kernel will send packets to the specific hardware queue.

To verify:

See individual queue statistics in the output of 'ethtool -S ' command.

Parameters for QoS Scheduling Configuration

The following shows sample scheduling configuration for hardware queues on the compute node.

The priority group ID and the corresponding scheduling algorithm and bandwidth to be used by the priority group can be configured.

Possible values for the scheduling algorithm include:

- strict
- rr (round-robin)

When round-robin scheduling is used, the percentage of total hardware queue bandwidth that can be used by the priority group is specified in the bandwidth parameter.

The following configuration and provisioning is applicable only for compute nodes running Niantic NICs and running kernel based vrouter.

```
qos_niantic = {
    'compute1': [
        { 'priority_id': '1', 'scheduling': 'strict', 'bandwidth': '0'},
        { 'priority_id': '2', 'scheduling': 'rr', 'bandwidth': '20'},
        { 'priority_id': '3', 'scheduling': 'rr', 'bandwidth': '10'}
    ],
    'compute2' :[
```

```

        { 'priority_id': '1', 'scheduling': 'strict', 'bandwidth': '0'},
        { 'priority_id': '1', 'scheduling': 'rr', 'bandwidth': '30'}
    ]
}

```

RELATED DOCUMENTATION

[Configuring Network QoS Parameters | 77](#)

<https://github.com/Juniper/contrail-controller/wiki/QoS>

Configuring Network QoS Parameters

IN THIS SECTION

- [Overview | 77](#)
- [QoS Configuration Examples | 78](#)
- [Limitations | 79](#)

Overview

You can use the OpenStack Nova command-line interface (CLI) to specify a quality of service (QoS) setting for a virtual machine's network interface, by setting the quota of a Nova flavor. Any virtual machine created with that Nova flavor will inherit all of the specified QoS settings. Additionally, if the virtual machine that was created with the QoS settings has multiple interfaces in different virtual networks, the same QoS settings will be applied to all of the network interfaces associated with the virtual machine. The QoS settings can be specified in unidirectional or bidirectional mode.

The quota driver in Neutron converts QoS parameters into libvirt network settings of the virtual machine.

The QoS parameters available in the quota driver only cover rate limiting the network interface. There are no specifications available for policy-based QoS at this time.

QoS Configuration Examples

Although the QoS setting can be specified in quota by using either Horizon or CLI, quota creation using CLI is more robust and stable, therefore, creating by CLI is the recommended method.

Example

CLI for Nova flavor has the following format:

```
nova flavor-key <flavor_name> set quota:vif_<direction>_<param_name> = value
```

where:

<flavor_name> is the name of an existing Nova flavor.

vif_<direction>_<param_name> is the inbound or outbound QoS data name.

QoS vif types include the following:

- vif_inbound_average lets you specify the average rate of inbound (receive) traffic, in kilobytes/sec.
- vif_outbound_average lets you specify the average rate of outbound (transmit) traffic, in kilobytes/sec.
- Optional: vif_inbound_peak and vif_outbound_peak specify the maximum rate of inbound and outbound traffic, respectively, in kilobytes/sec.
- Optional: vif_inbound_burst and vif_outbound_peak specify the amount of kilobytes that can be received or transmitted, respectively, in a single burst at the peak rate.

Details for various QoS parameters for libvirt can be found at <http://libvirt.org/formatnetwork.html>.

The following example shows an inbound average of 800 kilobytes/sec, a peak of 1000 kilobytes/sec, and a burst amount of 30 kilobytes.

```
nova flavor-key m1.small set quota:vif_inbound_average=800
nova flavor-key m1.small set quota:vif_inbound_peak=1000
nova flavor-key m1.small set quota:vif_inbound_burst=30
```

The following is an example of specified outbound parameters:

```
nova flavor-key m1.small set quota:vif_outbound_average=800
nova flavor-key m1.small set quota:vif_outbound_peak=1000
nova flavor-key m1.small set quota:vif_outbound_burst=30
```

After the Nova flavor is configured for QoS, a virtual machine instance can be created, using either Horizon or CLI. The instance will have network settings corresponding to the nova flavor-key, as in the following:

```
<interface type="ethernet">
  <mac address="02:a3:a0:87:7f:61"/>
  <model type="virtio"/>
  <script path=""/>
  <target dev="tapa3a0877f-61"/>
  <bandwidth>
    <inbound average="800" peak="1000" burst="30"/>
    <outbound average="800" peak="1000" burst="30"/>
  </bandwidth>
</interface>
```

Limitations

- The stock libvirt does not support rate limiting of ethernet interface types. Consequently, settings like those in the example for the guest interface will not result in any tc qdisc settings for the corresponding tap device in the host.
- The nova flavor-key `rxtx_factor` takes a float as an input and acts as a scaling factor for receive (inbound) and transmit (outbound) throughputs. This key is only available to Neutron extensions (private extensions). The Contrail Neutron plugin doesn't implement this private extension. Consequently, setting the nova flavor-key `rxtx_factor` will not have any effect on the QoS setting of the network interface(s) of any virtual machine created with that nova flavor.
- The outbound rate limits of a virtual machine interface are not strictly achieved. The outbound throughput of a virtual machine network interface is always less than the average outbound limit specified in the virtual machine's libvirt configuration file. The same behavior is also seen when using a Linux bridge.

RELATED DOCUMENTATION

| [Quality of Service in Contrail](#) | 69

CHAPTER 6

Load Balancers

IN THIS CHAPTER

- Using Load Balancers in Contrail | 80
- Support for OpenStack LBaaS Version 2.0 APIs | 94
- Configuring Load Balancing as a Service in Contrail | 97

Using Load Balancers in Contrail

IN THIS SECTION

- Invoking LBaaS Drivers | 80
- Benefits of Creating Configuration Objects | 81
- Using a Service Appliance Set as the LBaaS Provider | 82
- Understanding the Load Balancer Agent | 84
- F5 Networks Load Balancer Integration in Contrail | 84
- Example: Creating a Load Balancer | 87
- Using the Avi Networks Load Balancer for Contrail | 88

As of Contrail Release 3.0, load balancer LBaaS features are available. This topic includes:

Invoking LBaaS Drivers

The provider field specified in the pool configuration determines which load balancer drivers are selected. The load balancer driver selected is responsible for configuring the external hardware or virtual machine load balancer.

Supported load balancer drivers include:

- HAProxy
- A10 Networks
- F5 Networks
- Avi Networks

Benefits of Creating Configuration Objects

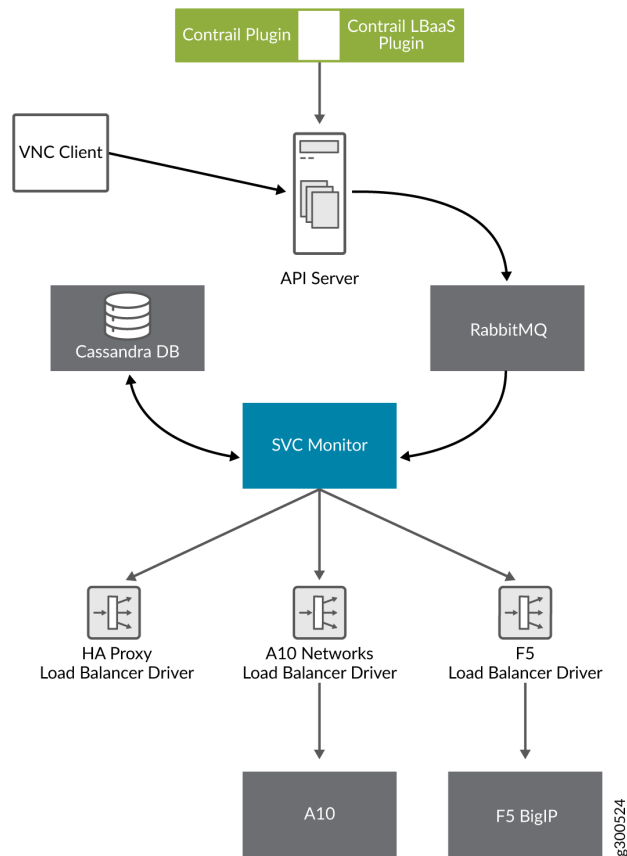
Starting with Contrail 3.0, the Neutron LBaaS plugin creates required configuration objects (such as pool, VIP, members, and monitor) in the Contrail API server, instead of within the Neutron plugin context, as in previous releases.

This method of configuration has the following benefits:

- Configuration objects can be created in multiple ways: from Neutron, from virtual controller APIs, or from the Contrail UI.
- The load balancer driver can make inline calls, such as REST or SUDS, to configure the external load balancer device.
- The load balancer driver can use Contrail service monitor infrastructure, such as database, logging, and API server.

[Figure 20 on page 82](#) provides an overview of the Contrail LBaaS components.

Figure 20: Contrail LBaaS components



Using a Service Appliance Set as the LBaaS Provider

In OpenStack Neutron, the load balancer provider is statically configured in `neutron.conf`, which requires restart of the Neutron server when configuring a new provider. The following is an example of the service provider configuration in `neutron.conf`.

```
[service_providers]
service_provider = LOADBALANCER:Opencontrail:neutron_plugin_contrail.plugins.opencontrail.
loadbalancer.driver.OpencontrailLoadbalancerDriver:default
```

In Contrail Release 3.0 and greater, the Neutron LBaaS provider is configured by using the object `service-appliance-set`. All of the configuration parameters of the LBaaS driver are populated to the `service-appliance-set` object and passed to the driver.

During initialization, the service monitor creates a default service appliance set with a default LBaaS provider, which uses an HAProxy-based load balancer. The service appliance set consists of individual

service appliances for load balancing the traffic. The service appliances can be physical devices or virtual machines.

Sample Configuration: Service Appliance Set

The following is a sample configuration of the service appliance set for the LBaaS provider:

```
{
  "service-appliance-set": {
    "fq_name": [
      "default-global-system-config",
      "f5"
    ],
    "service_appliance_driver":
      "svc_monitor.services.loadbalancer.drivers.f5.f5_driver.OpencontrailF5LoadbalancerDriver",
    "parent_type": "global-system-config",
    "service_appliance_set_properties": {
      "key_value_pair": [
        {
          "key": "sync_mode",
          "value": "replication"
        },
        {
          "key": "global_routed_mode",
          "value": "True"
        }
      ]
    },
    "name": "f5"
  }
}
```

Sample Configuration: Single Service Appliance

The following is a sample configuration of a single service appliance:

```
{
  "service-appliance": {
    "fq_name": [
      "default-global-system-config",
      "f5",
      "bigip"
    ],
  },
}
```

```

    "parent_type": "service-appliance-set",
    "service_appliance_ip_address": "<ip address>",
    "service_appliance_user_credentials": {
        "username": "admin",
        "password": "<password>"
    },
    "name": "bigip"
}
}

```

Understanding the Load Balancer Agent

The load balancer agent is a module in the service monitor. The service monitor listens on the RabbitMQ configuration messaging queue (`vnc_config.object-update`) to get configuration objects. The dependency tracker triggers changes to all related objects, based on configuration updates.

The dependency tracker is informed to notify the pool object whenever the VIP, member, or health monitor object is modified.

Whenever there is an update to the pool object, either directly due to a pool update or due to a dependency update, the load balancer agent in the service monitor is notified.

The load balancer agent module handles the following:

- Loading and unloading LBaaS driver-based service appliance set configuration.
- Providing the abstract driver class for the load balancer driver.
- Invoking the LBaaS driver.
- Load balancer-related configuration.

F5 Networks Load Balancer Integration in Contrail

This section details use of the F5 load balancer driver with Contrail.

Contrail Release 3.0 implements an LBaaS driver that supports a physical or virtual F5 Networks load balancer, using the abstract load balancer driver class, `ContrailLoadBalancerAbstractDriver`.

This driver is invoked from the load balancer agent of the `contrail-svc-monitor`. The driver makes a BIG-IP interface call to configure the F5 Networks device. All of the configuration parameters used to tune the driver are configured in the `service-appliance-set` object and passed to the driver by the load balancer agent while loading the driver.

The F5 load balancer driver uses the BIG-IP interface version V1.0.6, which is a Python package extracted from the load balancer plugin provided by F5 Networks. The driver uses either a SOAP API or a REST API.

F5 Load Balancer Global Routed Mode

The F5 load balancer driver is programmed in `global routed mode` using a property of the `service-appliance-set`.

This section describes the features and requirements of the F5 load balancer driver configured in `global routed mode`.

The following are features of the `global routed mode`.

- All virtual IP addresses (VIPs) are assumed to be routable from clients and all members are routable from the F5 device.
- All access to and from the F5 device is assumed to be globally routed, with no segregation between tenant services on the F5 device. Consequently, do NOT configure overlapping addresses across tenants and networks.
- The F5 device can be attached to the corporate network or to the IP fabric.

The following are requirements to support `global routed mode` of an F5 device used with LBaaS:

- The entire configuration of the F5 device for Layer 2 and Layer 3 is preprovisioned.
- All tenant networks and all IP fabrics are in the same namespace as the corporate network.
- All VIPs are in the same namespace as the tenant and corporate networks.

Traffic Flow in Global Routed Mode

This section describes and illustrates the behavior of traffic flow in `global routed mode`.

The information in this section is based on a model that includes the following network topology:

Corporate Network --- DC Gateway (MX device) --- IP Fabric --- Compute nodes

The Corporate Network, the IP Fabric and all tenant networks use IP addresses from a single namespace, there is no overlap of the addresses in the networks. The F5 devices can be attached to the Corporate Network or to the IP Fabric, and are configured to use the `global routed mode`.

The role of the MX Series device is to route post-proxy traffic, coming from the F5 device in the underlay, to the pool members in the overlay. In the reverse direction, the MX device takes traffic coming from the pool members in the overlay and routes it back to the F5 device in the underlay.

The MX device is preprovisioned with the following:

- VRF connected to pool network 2
- ability to route traffic from inet.0 to the pool network

The MX routes the traffic from inet.0 to public VRF and sends traffic to the compute node where the pool member is instantiated.

The F5 device is preprovisioned with the following:

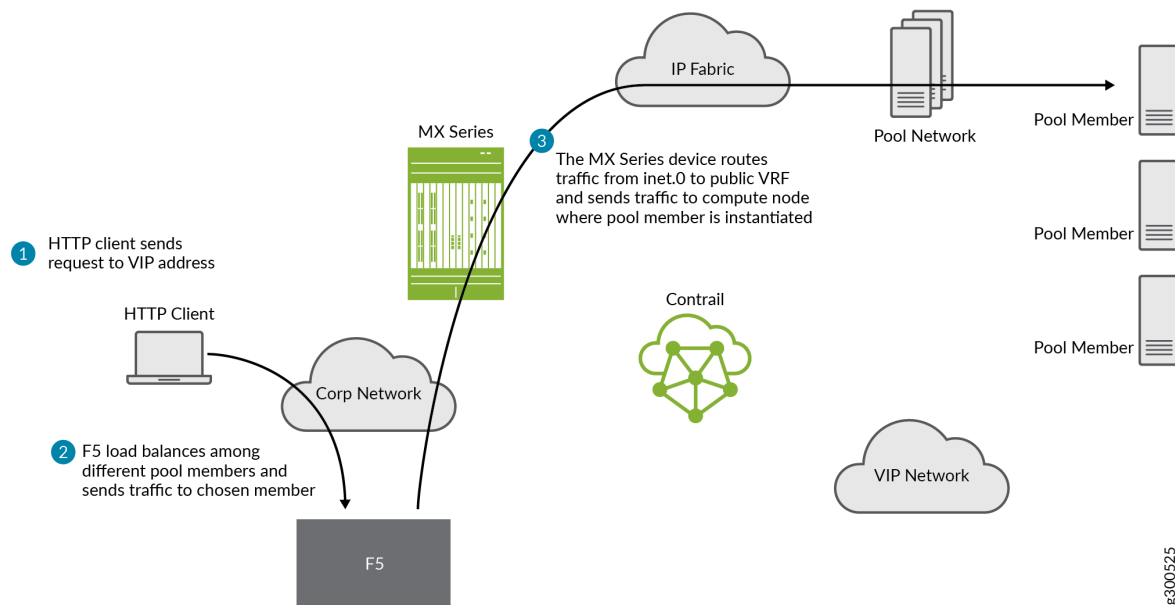
- publish route to attract VIP traffic
- pool network subnet route that points to the MX device

The F5 device is responsible for attracting traffic destined to all the VIPs, by advertising a subnet route that covers all VIPs using IGP.

The F5 device load balances among different pool members and sends traffic to the chosen member.

Figure 21 on page 86 shows the traffic flow in global routed mode.

Figure 21: Global Routed Traffic Flow



A similar result can also be achieved on the switch to which the F5 is attached, by publishing the VIP subnet in IGP and using a static route to point the VIP traffic to the F5 device.

The MX should attract the reverse traffic from the pool members going back to the F5.

Routing Traffic to Pool Members

For post load balancing traffic going from the F5 device to the pool members, the MX Series device needs to attract traffic for all the tenant networks.

Routing Reverse Traffic from Pool Members to the F5 Device

The MX should attract the reverse traffic from the pool members going back to the F5.

Initial Configuration on an F5 Device

- The operator is responsible for ensuring that the F5 device attracts traffic to all VIP subnets by injecting the route for the VIP subnet into IGP. Alternately, the switch to which F5 is connected can advertise the VIP subnet route and use the static route to send VIP traffic to the F5 device.
- In the global routed mode, the F5 uses AutoMap SNAT for all VIP traffic.

Initial Configuration on an MX Series Device Used as DC Gateway

- The operator must identify a super-net that contains all tenant network subnets (pool members across multiple pools) and advertise its route into corporate and fabric networks, using IGP (preferred) or static routes.
- The operator must add a static route for the super-net into inet.0 with a next-hop of public.inet.0.
- The operator must create a public VRF and get its default route imported into the VRF. This is to attract the return traffic from pool members to the F5 device (VIP destination).

Configuration on MX Device for Each Pool Member

- For each member virtual network, the operator adds a policy to connect the member pool virtual network to the public virtual network.
- As new member virtual networks are connected to the public virtual network by policy, corresponding targets are imported by the public VRF on MX. The Contrail Device Manager generates the configuration of import, export targets for public VRF on the MX device.
- The operator must ensure that security group rules for the member virtual network ports allow traffic coming from the F5 device.

Example: Creating a Load Balancer

Use the following steps to create a load balancer in Contrail Release 3.0 and greater.

1. To configure a service appliance set, use the script in `/opt/contrail/utils` to create a load balancer provider. With the script, you specify the driver and name of the selected provider. Additional configuration can be performed using the key-value pair property configuration.

```
/opt/contrail/utils/service_appliance_set.py --api_server_ip <ip address>--api_server_port 8082 --oper add --
admin_user admin --admin_password <password> --admin_tenant_name admin --name f5 --driver
"svc_monitor.services.loadbalancer.drivers.f5.f5_driver.OpencontrailF5LoadbalancerDriver" --properties
'{"use_snat": "True", "num_snat": "1", "global_routed_mode":"True", "sync_mode": "replication", "vip_vlan":
"trial2"}'
```

2. Add the actual device information of the load balancer.

```
/opt/contrail/utils/service_appliance.py --api_server_ip <ip address>--api_server_port 8082 --oper add --
admin_user admin --admin_password <password> --admin_tenant_name admin --name bigip --service_appliance_set f5
--device_ip 10.204.216.113 --user_credential '{"user": "admin", "password": "<password>"}'
```

3. Refer to the load balancer provider while configuring the pool.

```
neutron lb-pool-create --lb-method ROUND_ROBIN --name web_service --protocol HTTP --provider "f5" --subnet-id
<subnet id>
```

4. Add members to the load balancer pool. Both bare metal webserver and overlay webserver are allowed as pool members. The F5 device can load balance the traffic among all pool members.

```
neutron lb-member-create --address <ip address>--protocol-port 8080 --weight 3 web_service

neutron lb-member-create --address <ip address> --protocol-port 8080 --weight 2 web_service
```

5. Create a VIP for the load balancer pool.

```
neutron lb-vip-create --name httpserver --protocol-port 80 --protocol HTTP web_service --subnet-id <subnet id>
```

6. Create the health monitor and associate it with the load balancer pool.

```
neutron lb-healthmonitor-create --delay 3 --type HTTP --max-retries 3 --timeout 3

neutron lb-healthmonitor-associate <nnnnn-nnnnn-nnnn-> web_service
```

Using the Avi Networks Load Balancer for Contrail

If you are using the Avi LBaaS driver in an OpenStack Contrail environment, there are two possible modes that are mutually-exclusive. The Avi Vantage cloud configuration is exactly the same in both modes:

- Neutron-based Avi LBaaS driver

In this mode, the Avi LBaaS driver derives from Neutron and resides in the Neutron server process. This mode enables coexistence of multiple Neutron LBaaS providers.

- Contrail-based Avi LBaaS driver

In this mode, the Avi LBaaS driver derives from Contrail and resides in the service-monitor process. This mode enables coexistence of multiple Contrail LBaaS providers.

NOTE: In a Contrail environment, you cannot have a mix of Contrail LBaaS and Neutron LBaaS. You must select a mode that is compatible with the current environment.

Installing the Avi LBaaS Neutron Driver

Use the following procedure to install the Avi Networks LBaaS load balancer driver for the Neutron server for Contrail.

The following steps are performed on the Neutron server host.

1. Determine the installed version of the Contrail Neutron plugin.

```
$ contrail-version neutron-plugin-contrail
Package Version
-----
neutron-plugin-contrail 3.0.2.0-51
```

2. Adjust the `neutron.conf` database connection URL.

```
$ vi /etc/neutron/neutron.conf
# if using mysql
connection = mysql+pymysql://neutron:c0ntrail123@127.0.0.1/neutron
```

3. Populate and upgrade the Neutron database schema.

```
# to upgrade to head
$ neutron-db-manage upgrade head
# to upgrade to a specific version
$ neutron-db-manage --config-file /etc/neutron/neutron.conf upgrade liberty
```

4. Drop foreign key constraints.

```
# obtain current mysql token
$ cat /etc/contrail/mysql.token
```



```
fabe17d9dd5ae798f7ea

$ mysql -u root -p
Enter password: fabe17d9dd5ae798f7ea

mysql> use neutron;

mysql> show create table vips;
# CONSTRAINT `vips_ibfk_1` FOREIGN KEY (`port_id`) REFERENCES `ports` (`id`) - ports table is
not used by Contrail
mysql> alter table vips drop FOREIGN KEY vips_ibfk_1;

mysql> show create table lbaas_loadbalancers;
# CONSTRAINT `fk_lbaas_loadbalancers_ports_id` FOREIGN KEY (`vip_port_id`) REFERENCES `ports`
(`id`)
mysql> alter table lbaas_loadbalancers drop FOREIGN KEY fk_lbaas_loadbalancers_ports_id;
```

5. To install the Avi LBaaS plugin, continue with steps from the readme file that downloads with the Avi LBaaS software. You can perform either a local installation or a manual installation. The following are sample installation steps.

- For a local installation:

```
# LBaaS v1 driver
$ ./install.sh --aname avi_adc --aip

<controller_ip|controller_vip>
--auser

--apass

# LBaaS v2 driver
$ ./install.sh --aname avi_adc_v2 --aip
<controller_ip|controller_vip>
--auser

--apass

--v2
```

- For a manual installation:

```
# LBaaS v1 driver
$ vi /etc/neutron/neutron.conf
#service_plugins =
neutron_plugin_contrail.plugins.opencontrail.loadbalancer.plugin.LoadBalancerPlugin
service_plugins = neutron_lbaas.services.loadbalancer.plugin.LoadBalancerPlugin
[service_providers]
service_provider =
LOADBALANCER:Avi_ADC:neutron_lbaas.services.loadbalancer.drivers.avi.avi_driver.AviLbaaSDriver

[avi_adc]
address=10.1.11.4
user=admin
password=avi123
cloud=jcos

# LBaaS v2 driver
$ vi /etc/neutron/neutron.conf
#service_plugins =
neutron_plugin_contrail.plugins.opencontrail.loadbalancer.plugin.LoadBalancerPlugin
service_plugins = neutron_lbaas.services.loadbalancer.plugin.LoadBalancerPluginv2
[service_providers]
service_provider = LOADBALANCERV2:avi_adc_v2:neutron_lbaas.drivers.avi.driver.AviDriver

[avi_adc_v2]
controller_ip=10.1.11.3
username=admin
password=avi123

$ service neutron-server restart
$ neutron service-provider-list
```

Installing the Avi LBaaS Contrail Driver

Use the following procedure to install the Avi Networks LBaaS load balancer driver for Contrail.

The following steps are performed on the Contrail api-server host.

1. Determine the installed version of the Contrail Neutron plugin.

```
$ contrail-version neutron-plugin-contrail
Package Version
-----
neutron-plugin-contrail 3.0.2.0-51
```

2. Install the Avi driver.

```
# LBaaS v2 driver
$ ./install.sh --aname ocavi_adc_v2 --aip

<controller_ip|controller_vip>
--auser

--apass

--v2 --no-restart --no-confmodify
```

3. Set up the service appliance set.

NOTE: If `neutron_lbaas` doesn't exist on the `api-server` node, adjust the driver path to the correct path location for `neutron_lbaas`.

```
$ /opt/contrail/utils/service_appliance_set.py --api_server_ip 10.xx.xx.100 --api_server_port 8082 --oper add
--admin_user admin --admin_password <password> --admin_tenant_name admin --name ocavi_adc_v2 --driver
"neutron_lbaas.drivers.avi.avi_ocdriver.OpencontrailAviLoadbalancerDriver" --properties '{"address":
"10.1.xx.3", "user": "admin", "password": "avi123", "cloud": "Default-Cloud"}'
```

4. To delete the service appliance set.

```
$ /opt/contrail/utils/service_appliance_set.py --api_server_ip 10.xx.xx.100 --api_server_port 8082 --oper del
--admin_user admin --admin_password <password> --admin_tenant_name admin --name ocavi_adc_v2
```

Configuring the Avi Controller

1. If OpenStack endpoints are private IPs and Contrail provides a public front-end IP to those endpoints, use iptables to DNAT. On the AviController only, perform iptable NAT to reach the private IPs.

```
$ iptables -t nat -I OUTPUT --dest 17x.xx.xx.50 -j DNAT --to-dest 10.xx.xx.100
```

2. To configure the Avi controller during cloud configuration, select the “Integration with Contrail” checkbox and provide the endpoint URL of the Contrail VNC api-server. Use the Keystone credentials from the OpenStack configuration to authenticate with the api-server service.

Example Configuration Settings

```
: > show cloud jcos
```

Field	Value
uuid	cloud-104bb7e6-a9d2-4b34-a4c5-d94be659bb91
name	jcos
vtype	CLOUD_OPENSTACK
openstack_configuration	
username	admin
admin_tenant	demo
keystone_host	17x.xx.xx.50
mgmt_network_name	mgmtnw
privilege	WRITE_ACCESS
use_keystone_auth	True
region	RegionOne
hypervisor	KVM
tenant_se	True
import_keystone_tenants	True
anti_affinity	True
port_security	False
security_groups	True
allowed_address_pairs	True
free_floatingips	True
img_format	OS_IMG_FMT_AUTO
use_admin_url	True
use_internal_endpoints	False
config_drive	True
insecure	True
intf_sec_ips	False
external_networks	False
neutron_rbac	True
nuage_port	8443
contrail_endpoint	http://10.10.10.100:8082
apic_mode	False
dhcp_enabled	True
mtu	1500 bytes

	prefer_static_routes		False	
	enable_vip_static_routes		False	
	license_type		LIC_CORES	
	tenant_ref		admin	
+-----+-----+				

RELATED DOCUMENTATION

	Configuring Load Balancing as a Service in Contrail 97
	Support for OpenStack LBaaS Version 2.0 APIs 94

Support for OpenStack LBaaS Version 2.0 APIs

IN THIS SECTION

- [Platform Support | 94](#)
- [Using OpenStack LBaaS Version 2.0 | 95](#)
- [Support for Multiple Certificates per Listener | 96](#)
- [Neutron Load-Balancer Creation | 96](#)

Starting with Release 3.1, Contrail provides support for the OpenStack Load Balancer as a Service (LBaaS) Version 2.0 APIs in the Liberty release of OpenStack.

Platform Support

Table 9 on page 94 shows which Contrail with OpenStack release combinations support which version of OpenStack LBaaS APIs.

Table 9: Contrail OpenStack Platform Support for LBaaS Versions

Contrail OpenStack Platform	LBaaS Support
Contrail-3.1-Liberty (and subsequent OS releases)	Only LBaaS v2 is supported.

Table 9: Contrail OpenStack Platform Support for LBaaS Versions (Continued)

Contrail OpenStack Platform	LBaaS Support
Contrail-3.0-Liberty (and subsequent OS releases)	LBaaS v1 is default. LBaaS v2 is Beta.
<Contrail-any-release>-Kilo (and previous OS releases)	Only LBaaS v1 is supported.

Using OpenStack LBaaS Version 2.0

The OpenStack LBaaS Version 2.0 extension enables tenants to manage load balancers for VMs, for example, load-balancing client traffic from a network to application services, such as VMs, on the same network. The LBaaS Version 2.0 extension is used to create and manage load balancers, listeners, pools, members of a pool, and health monitors, and to view the status of a resource.

For LBaaS v2.0, the Contrail controller aggregates the configuration by provider. For example, if haproxy is the provider, the controller generates the configuration for haproxy and eliminates the need to send all of the load-balancer resources to the vrouter-agent; only the generated configuration is sent, as part of the service instance.

For more information about OpenStack v2.0 APIs, refer to the section *LBaaS 2.0 (STABLE) (lbaas, loadbalancers, listeners, health_monitors, pools, members)*, at <http://developer.openstack.org/api-ref-networking-v2-ext.html>.

LBaaS v2.0 also allows users to listen to multiple ports for the same virtual IP, by decoupling the virtual IP address from the port.

The object model has the following resources:

- Load balancer—Holds the virtual IP address
- Listeners—One or many listeners with different ports, protocols, and so on
- Pools
- Members
- Health monitors

Support for Multiple Certificates per Listener

Multiple certificates per listener are supported, with OpenStack Barbican as the storage for certificates. OpenStack Barbican is a REST API designed for the secure storage, provisioning, and management of secrets such as passwords, encryption keys, and X.509 certificates.

The following is an example CLI to store certificates in Barbican:

```
- barbican --os-identity-api-version 2.0 secret store --payload-content-type='text/plain' --name='certificate' --payload="$(cat server.crt)"
```

For more information about OpenStack Barbican, see: <https://wiki.openstack.org/wiki/Barbican>.

Neutron Load-Balancer Creation

The following is an example of Neutron load-balancer creation:

```
- neutron net-create private-net

- neutron subnet-create --name private-subnet private-net 10.30.30.0/24

- neutron lbaas-loadbalancer-create $(neutron subnet-list | awk '/ private-subnet / {print $2}')
--name lb1

- neutron lbaas-listener-create --loadbalancer lb1 --protocol-port 443 --protocol
TERMINATED_HTTPS --name listener1 --default-tls-container=$(barbican --os-identity-api-version
2.0 container list | awk '/ tls_container / {print $2}')

- neutron lbaas-pool-create --name pool1 --protocol HTTP --listener listener1 --lb-algorithm
ROUND_ROBIN

- neutron lbaas-member-create --subnet private-subnet --address 30.30.30.10 --protocol-port 80
mypool

- neutron lbaas-member-create --subnet private-subnet --address 30.30.30.11 --protocol-port 80
mypool
```

RELATED DOCUMENTATION

<https://wiki.openstack.org/wiki/Barbican>

<http://developer.openstack.org/api-ref-networking-v2-ext.html>

Configuring Load Balancing as a Service in Contrail

IN THIS SECTION

- [Overview: Load Balancing as a Service | 97](#)
- [Contrail LBaaS Implementation | 98](#)
- [Configuring LBaaS Using CLI | 99](#)
- [Configuring LBaaS using the Contrail Web UI | 101](#)

Overview: Load Balancing as a Service

NOTE: The *Neutron* LBaaS plugin is not supported in OpenStack Train release. From OpenStack Train release, *neutron-lbaas* is replaced by *Octavia*.

Load Balancing as a Service (LBaaS) is a feature available through OpenStack Neutron. Contrail Release 1.20 and greater allows the use of the Neutron API for LBaaS to apply open source load balancing technologies to provision a load balancer in the Contrail system.

The LBaaS load balancer enables the creation of a pool of virtual machines serving applications, all front-ended by a virtual-ip. The LBaaS implementation has the following features:

- Load balancing of traffic from clients to a pool of backend servers. The load balancer proxies all connections to its virtual IP.
- Provides load balancing for HTTP, TCP, and HTTPS.
- Provides health monitoring capabilities for applications, including HTTP, TCP, and ping.
- Enables floating IP association to virtual-ip for public access to the backend pool.

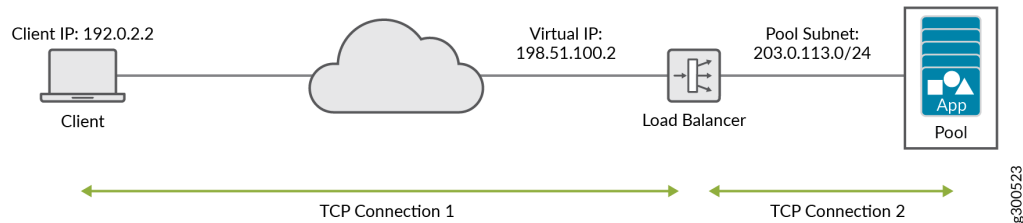
In [Figure 22 on page 98](#), the load balancer is launched with the virtual IP address 198.51.100.2. The backend pool of virtual machine applications (App Pool) is on the subnet 203.0.113.0/24. Each of the application virtual machines gets an IP address (virtual-ip) from the pool subnet. When a client connects

to the virtual-ip for accessing the application, the load balancer proxies the TCP connection on its virtual-ip, then creates a new TCP connection to one of the virtual machines in the pool.

The pool member is selected using one of following methods:

- weighted round robin (WRR), based on the weight assignment
- least connection, selects the member with the fewest connections
- source IP selects based on the source-ip of the packet

Figure 22: Load Balancing as a Service in Contrail



Additionally, the load balancer monitors the health of each pool member using the following methods:

- Monitors TCP by creating a TCP connection at intervals.
- Monitors HTTP by creating a TCP connection and issuing an HTTP request at intervals.
- Monitors ping by checking if a member can be reached by pinging.

Contrail LBaaS Implementation

Contrail supports the OpenStack LBaaS Neutron APIs and creates relevant objects for LBaaS, including virtual-ip, loadbalancer-pool, loadbalancer-member, and loadbalancer-healthmonitor. Contrail creates a service instance when a loadbalancer-pool is associated with a virtual-ip object. The service scheduler then launches a namespace on a randomly selected virtual router and spawns HAProxy into that namespace. The configuration for HAProxy is picked up from the load balancer objects. Contrail supports high availability of namespaces and HAProxy by spawning active and standby on two different routers.

A Note on Installation

To use the LBaaS feature, HAProxy, version 1.5 or greater and iproute2, version 3.10.0 or greater must both be installed on the Contrail compute nodes.

If you are using fab commands for installation, the haproxy and iproute2 packages will be installed automatically with LBaaS if you set the following:

```
env.enable_lbaas=True
```

Use the following to check the version of the iproute2 package on your system:

```
root@nodeh5:/var/log# ip -V
ip utility, iproute2-ss130716
root@nodeh5:/var/log#
```

Limitations

LBaaS currently has these limitations:

- A pool should not be deleted before deleting the VIP.
- Multiple VIPs cannot be associated with the same pool. If pool needs to be reused, create another pool with the same members and bind it to the second VIP.
- Members cannot be moved from one pool to another. If needed, first delete the members from one pool, then add to a different pool.
- In case of active-standby failover, namespaces might not get cleaned up when the agent restarts.
- The floating-ip association needs to select the VIP port and not the service ports.

Configuring LBaaS Using CLI

The LBaaS feature is enabled on Contrail through Neutron API calls. The following procedure shows how to create a pool network and a VIP network using CLI. The VIP network is created in the public network and members are added in the pool network.

Creating a Load Balancer

Use the following steps to create a load balancer in Contrail.

1. Create a VIP network.

```
neutron net-create vipnet

neutron subnet-create --name vipsubnet vipnet 198.51.100.2
```

2. Create a pool network.

```
neutron net-create poolnet
```

```
neutron subnet-create --name poolsubnet poolnet 203.0.113.0/24
```

3. Create a pool for HTTP.

```
neutron lb-pool-create --lb-method ROUND_ROBIN --name mypool --protocol HTTP --subnet-id poolsubnet
```

4. Add members to the pool.

```
neutron lb-member-create --address 203.0.113.3 --protocol-port 80 mypool
```

```
neutron lb-member-create --address 203.0.113.4 --protocol-port 80 mypool
```

5. Create a VIP for HTTP and associate it to the pool.

```
neutron lb-vip-create --name myvip --protocol-port 80 --protocol HTTP --subnet-id vipsubnet mypool
```

Deleting a Load Balancer

Use the following steps to delete a load balancer in Contrail.

1. Delete the VIP.

```
neutron lb-vip-delete <vip-uuid>
```

2. Delete members from the pool.

```
neutron lb-member-delete <member-uuid>
```

3. Delete the pool.

```
neutron lb-pool-delete <pool-uuid>
```

Managing Healthmonitor for Load Balancer

Use the following commands to create a healthmonitor, associate a healthmonitor to a pool, disassociate a healthmonitor, and delete a healthmonitor.

1. Create a healthmonitor.

```
neutron lb-healthmonitor-create --delay 20 --timeout 10 --max-retries 3 --type HTTP
```

2. Associate a healthmonitor to a pool.

```
neutron lb-healthmonitor-associate <healthmonitor-uuid> mypool
```

3. Disassociate a healthmonitor from a pool.

```
neutron lb-healthmonitor-disassociate <healthmonitor-uuid> mypool
```

Configuring an SSL VIP with an HTTP Backend Pool

Use the following steps to configure an SSL VIP with an HTTP backend pool.

1. Copy an SSL certificate to all compute nodes.

```
scp ssl_certificate.pem <compute-node-ip> <certificate-path>
```

2. Update the information in `/etc/contrail/contrail-vrouter-agent.conf`.

```
# SSL certificate path haproxy

haproxy_ssl_cert_path=<certificate-path>
```

3. Restart contrail-vrouter-agent.

```
service contrail-vrouter-agent restart
```

4. Create a VIP for port 443 (SSL).

```
neutron lb-vip-create --name myvip --protocol-port 443 --protocol HTTP --subnet-id vipsubnet mypool
```

Configuring LBaaS using the Contrail Web UI

Create, edit, or delete load balancers using the Contrail Web UI. Use the following guidelines when creating load balancers:

- Each load balancer consists of one or more listeners, pools, pool members, and health monitors.
 - Listener: Port that listens for traffic from a particular load balancer. Multiple listeners can be associated with a single load balancer.
 - Pool: Group of hosts that serves traffic from the load balancer.
 - Pool Member: Server that is specified by the IP address and port for which it uses to serve the traffic it receives from the load balancer.
 - Health Monitor: Health monitors are associated with pools and help divert traffic away from pool members that are temporarily offline.
- Each load balancer can have multiple pools with one or more listeners for each pool.

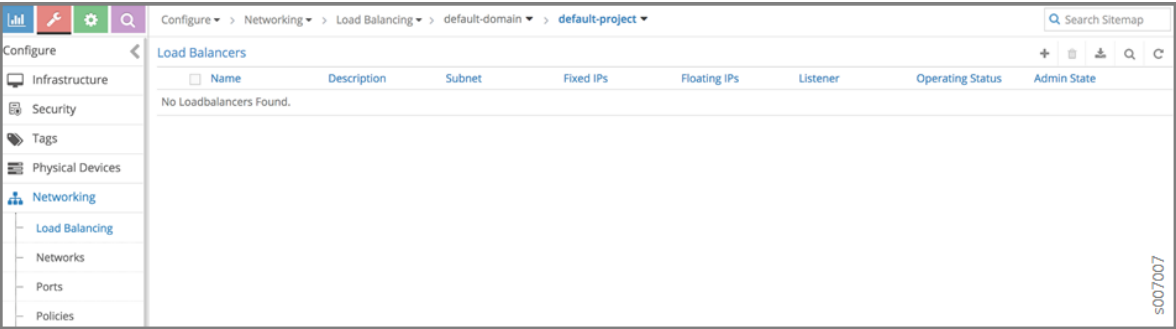
- The native load balancer has a single pool that is shared among multiple listeners.

Creating a Load Balancer

Use the following steps to create a load balancer with the load balancer wizard.

1. Go to **Configure > Networking > Load Balancing**. A summary screen of the Load Balancers is displayed; see [Figure 23 on page 102](#).

Figure 23: Summary Screen of Load Balancers




2. To create a load balancer, click the  icon on the Load Balancers summary screen. The first window of the Create Load Balancer wizard is displayed.

Figure 24: Load Balancer Information

Create Load Balancer

1 Load Balancer 2 Listener 3 Pool 4 Pool Member 5 Monitor

Name* LBaaS1

Description

Subnet* fb687efe-7e84-4b18-b719-f5733132a3b3

Fixed IPs xxx.xxx.xxx.xxx

Loadbalancer Provider* opencontrail

Floating IP Select Floating IP

Admin State ☒

Cancel Next

s007008

Add the load balancer information:

- **Name:** Name of the load balancer.
- **Description:** (Optional) Description of the load balancer.
- **Subnet:** Dropdown menu displays all subnets from list of all available networks. The subnet is the network on which to allocate the IP address of the load balancer.
- **Fixed IPs:** (Optional) IPv4 or IPv6 address.
- **Loadbalancer Provider:** Dropdown menu includes available options. Default is opencontrail.
- **Floating IP:** (Optional) IPv4 or IPv6 address.
- **Admin State:** Check the checkbox for UP or uncheck the checkbox for DOWN. Default is UP.

3. Click **Next**. The Listener fields are displayed.

Figure 25: Listener Information

Create Load Balancer

Progress: 1 (Load Balancer) | 2 (Listener) | 3 (Pool) | 4 (Pool Member) | 5 (Monitor)

Name*
Listner-http

Description

Protocol*
HTTP

Port*
80

Connection Limit
-1

Admin State
☒

Buttons: Cancel, Previous, Next

Vertical text on the right: s007009

Add the listener information:

- **Name:** Name of the listener.
- **Description:** (Optional) Description of the listener.
- **Protocol:** Dropdown menu includes HTTP, TCP, and TERMINATED_HTTPS protocols. TERMINATED_HTTPS is available only if the key-manager service is enabled and you have access to the lists of SSL certificates.
- **Port:** Must be an integer in the range of 1 to 65535.
- **Connection Limit:** (Optional) Default value is -1, indicating an infinite limit.
- **Admin State:** Check the checkbox for UP or uncheck the checkbox for DOWN. Default is UP.

4. Click **Next**. The Pool fields are displayed.

Figure 26: Pool Information

Create Load Balancer

Load Balancer Listener **Pool** Pool Member Monitor

Name*
Pool1

Description

Method*
LEAST_CONNECTIONS

Protocol*
HTTP

Session Persistence
Select Session Persistence

Admin State
☒

Global Custom Attribute

Default Custom Attribute

Cancel Previous Next

s007010

Add the pool information:


- **Name:** Name of the pool.
 - **Description:** (Optional) Description of the pool.
 - **Method:** Load balancing method used to distribute incoming requests. Dropdown menu includes LEAST_CONNECTIONS, ROUND_ROBIN, and SOURCE_IP.
 - **Protocol:** The protocol used by the pool and its members for the load balancer traffic. Dropdown menu includes TCP, HTTP, and HTTPS.
 - **Session Persistence:** (Optional) Default value is an empty dictionary.
 - **Admin State:** Check the checkbox for UP or uncheck the checkbox for DOWN. Default is UP.
5. Click **Next**. The list of available pool member instances are displayed. To add an external member, click the  icon. Each pool member must have a unique IP address and port combination.

Figure 27: Pool Member Information

Create Load Balancer

Load Balancer Listener Pool Pool Member Monitor

Name	Subnet	IP Address*	Port	Weight	Admin State	+

Cancel Previous Next

s007011

The pool member information includes:

- **Name:** Name of the pool member.
- **Subnet:** The subnet in which to access the member.
- **IP Address:** The IP address of the member that is used to receive traffic from the load balancer.
- **Port:** The port to which the member listens to receive traffic from the load balancer.
- **Weight:** The default value is 1.
- **Admin State:** Check the checkbox for UP or uncheck the checkbox for DOWN. Default is UP.

6. Click **Next**. The Monitor fields are displayed.

Figure 28: Health Monitor Information

working > Load Balancing > default-domain > admin

Create Load Balancer

Load Balancer Listener Pool Pool Member Monitor

Monitor Type* HTTP

Health check interval (sec)* 5

Retry count before markdown* 3

Timeout (sec)* 5

HTTP Method GET

Expected HTTP Status Code 200

URL Path /

Cancel Previous Create Load Balancer

Add the health monitor information:

- **Monitor Type:** Dropdown menu includes HTTP, HTTPS, PING, and TCP.
- **Health check interval (sec):** The time interval, in seconds, between each health check.
- **Retry count before markdown:** The maximum number of failed health checks before the state of a member is changed to OFFLINE.
- **Timeout (sec):** The maximum number of seconds allowed for any given health check to complete. The timeout value should always be less than the health check interval.
- **HTTP Method:** Required if monitor type is HTTP. Dropdown menu includes GET and HEAD. The default value is GET.
- **Expected HTTP Status Code:** Required if monitor type is HTTP. The default value is 200.
- **URL Path:** Required if monitor type is HTTP. The default value is “/.”
- **Admin State:** Check the checkbox for UP or uncheck the checkbox for DOWN. Default is UP.

Viewing or Editing Load Balancers

Use the following steps to view or edit existing load balancers.

1. Go to **Configure > Networking > Load Balancing**. A summary screen of the Load Balancers is displayed.

Figure 29: Summary Screen of Load Balancers

Name	Description	Subnet	Fixed IPs	Floating IPs	Listener	Operating Status	Admin State
LBaaS1	-	192.168.0.1/24	192.168.0.3	-	1	Online	Yes

2. To view or edit a load balancer, click the name of a load balancer listed in the summary screen. The Load Balancer Info window is displayed.

Figure 30: Load Balancer Info Window

Load Balancer Details	Value
Name	LBaaS1-0f64eaca-3137-4c77-8467-a837f9f049c7
Display Name	LBaaS1
Description	-
Provider	opencontrail
Provisioning Status	Active
Operating Status	Online
Admin State	Yes
Fixed IPs	192.168.0.3
Floating IPs	-
Listener Count	1
HA Mode	active-standby
Service Instance	0f64eaca-3137-4c77-8467-a837f9f049c7
Virtual Machine Interface	017f3375-277c-4609-8198-87c6b5da0952

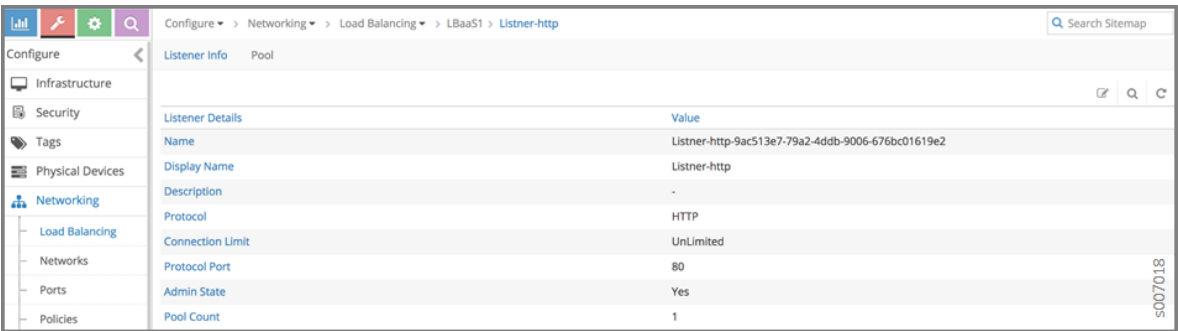
3. To view the list of listeners associated with the load balancer, click on the Listener tab. A summary screen of the listeners is displayed.

Figure 31: Summary Screen of Listeners



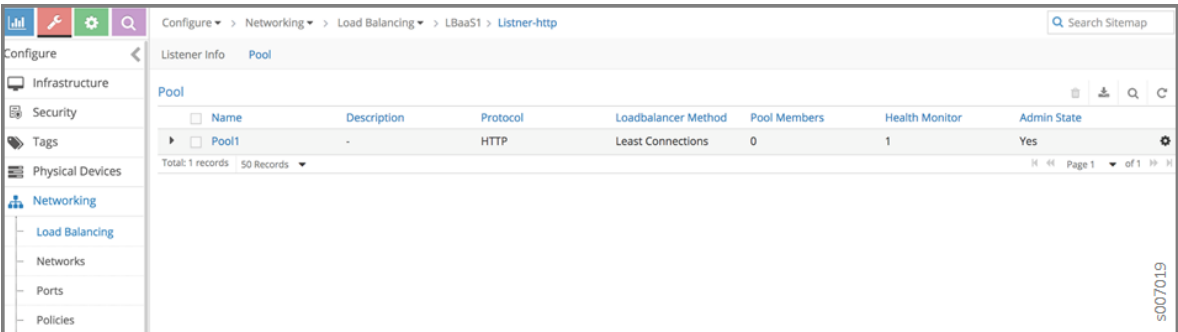
4. To view or edit a listener, click the name of a listener listed in the summary screen. The Listener Info window is displayed.

Figure 32: Listener Info Window



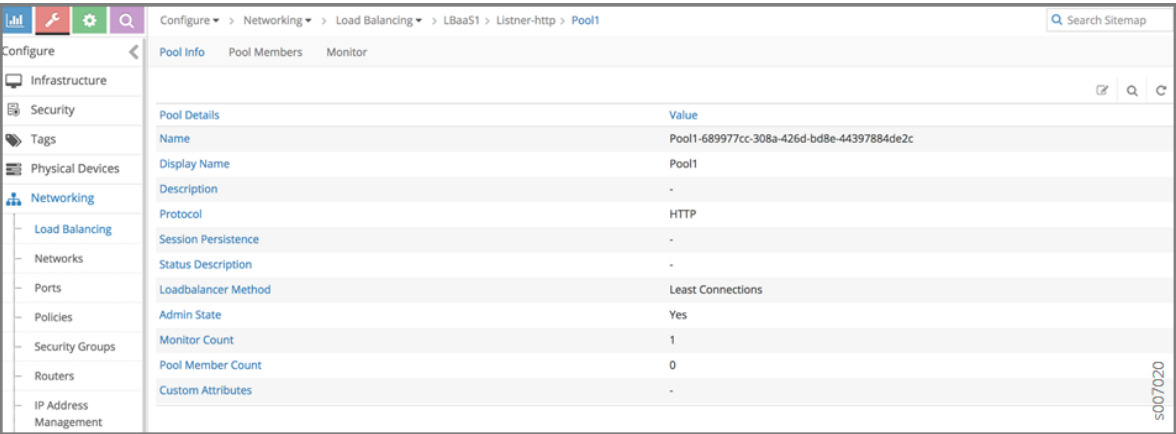
5. To view a list of pools associated with the listener, click on the Pool tab. A summary screen of the pools is displayed.

Figure 33: Summary Screen of Pools



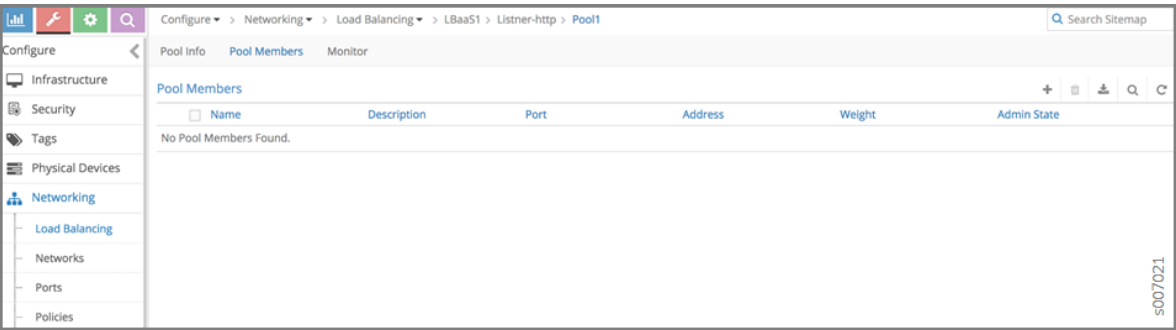
- 6. To view or edit a pool, click the name of a pool listed in the summary screen. The Pool Info window is displayed.

Figure 34: Pool Info Window



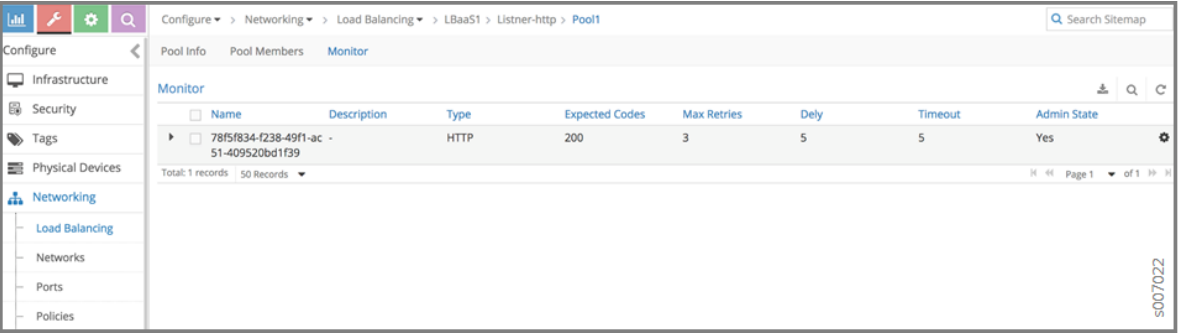
- 7. To view a list of members associated with the pool, click on the Pool Members tab. A summary screen of the pool members is displayed.

Figure 35: Pool Members Summary Screen



- 8. To view the health monitor details associated with the pool, click on the Monitor tab. The health monitor details are displayed.

Figure 36: Pool Members Summary Screen



Deleting a Load Balancer

Use the following steps to delete a load balancer.

1. Delete the members of the pools. Select the pool members you want to delete, then click the trashcan icon; see [Figure 35 on page 110](#).
2. Delete the pools. Select the pools you want to delete, then click the trashcan icon; see [Figure 33 on page 109](#).
3. Delete the listeners. Select the listeners you want to delete, then click the trashcan icon; see [Figure 31 on page 109](#).
4. Delete the load balancer. Select the load balancer you want to delete, then click the trashcan icon; see [Figure 29 on page 108](#).

RELATED DOCUMENTATION

Using Load Balancers in Contrail		80
Support for OpenStack LBaaS Version 2.0 APIs		94

Optimizing Contrail Networking

IN THIS CHAPTER

- [Multiqueue Virtio Interfaces in Virtual Machines | 112](#)

Multiqueue Virtio Interfaces in Virtual Machines

IN THIS SECTION

- [Multiqueue Virtio Overview | 112](#)
- [Requirements and Setup for Multiqueue Virtio Interfaces | 112](#)

Contrail 3.2 adds support for multiqueue for the DPDK-based router.

Contrail 3.1 supports multiqueue virtio interfaces for Ubuntu kernel-based router, only.

Multiqueue Virtio Overview

OpenStack Liberty supports the ability to create VMs with multiple queues on their virtio interfaces. Virtio is a Linux platform for I/O virtualization, providing a common set of I/O virtualization drivers. Multiqueue virtio is an approach that enables the processing of packet sending and receiving to be scaled to the number of available virtual CPUs (vCPUs) of a guest, through the use of multiple queues.

Requirements and Setup for Multiqueue Virtio Interfaces

To use multiqueue virtio interfaces, ensure your system meets the following requirements:

- The OpenStack version must be Liberty or greater.

- The maximum number of queues in the VM interface is set to the same value as the number of vCPUs in the guest.
- The VM image metadata property is set to enable multiple queues inside the VM.

Setting Virtual Machine Metadata for Multiple Queues

Use the following command on the OpenStack node to enable multiple queues on a VM:

```
source /etc/contrail/openstackrc  
nova image-meta <image_name> set hw_vif_multiqueue_enabled="true"
```

After the VM is spawned, use the following command on the virtio interface in the guest to enable multiple queues inside the VM:

```
ethtool -L <interface_name> combined <#queues>
```

Packets will now be forwarded on all queues in the VM to and from the vRouter running on the host.

NOTE: Multiple queues in the VM are only supported with the kernel mode vRouter in Contrail 3.1.

Contrail 3.2 adds support for multiple queues with the DPDK-based vrouter, using OpenStack Mitaka. The DPDK vrouter has the same setup requirements as the kernel mode vrouter. However, in the `ethtool -L` setup command, the number of queues cannot be higher than the number of CPU cores assigned to vrouter in the testbed file.

Contrail Networking OpenStack Analytics

IN THIS CHAPTER

- [Ceilometer Support in Contrail | 114](#)

Ceilometer Support in Contrail

IN THIS SECTION

- [Overview | 114](#)
- [Ceilometer Details | 115](#)
- [Verification of Ceilometer Operation | 115](#)
- [Contrail Ceilometer Plugin | 118](#)
- [Ceilometer Installation and Provisioning | 121](#)

Ceilometer is an OpenStack feature that provides an infrastructure for collecting SDN metrics from OpenStack projects. The metrics can be used by various rating engines to transform events into billable items. The Ceilometer collection process is sometimes referred to as “metering”. The Ceilometer service provides data that can be used by platforms that provide metering, tracking, billing, and similar services. This topic describes how to configure the Ceilometer service for Contrail.

Overview

Contrail Release 2.20 and later supports the OpenStack Ceilometer service, on the OpenStack Juno release on Ubuntu 14.04.1 LTS.

The prerequisites for installing Ceilometer are:

- Contrail Cloud installation

- Provisioned using `enable_ceilometer = True` in the `provisioning` file.

NOTE: Ceilometer services are only installed on the first OpenStack controller node and do not support high availability in Contrail Release 2.20.

Ceilometer Details

Ceilometer is used to reliably collect measurements of the utilization of the physical and virtual resources comprising deployed clouds, persist these data for subsequent retrieval and analysis, and trigger actions when defined criteria are met.

The Ceilometer architecture consists of:

Polling agent	Agent designed to poll OpenStack services and build meters. The polling agents are also run on the compute nodes in addition to the OpenStack controller.
Notification agent	Agent designed to listen to notifications on message queue and convert them to events and samples.
Collector	Gathers and records event and metering data created by the notification and polling agents.
API server	Provides a REST API to query and view data recorded by the collector service.
Alarms	Daemons to evaluate and notify based on defined alarming rules.
Database	Stores the metering data, notifications, and alarms. The supported databases are MongoDB, SQL-based databases compatible with SQLAlchemy, and HBase. The recommended database is MongoDB, which has been thoroughly tested with Contrail and deployed on a production scale.

Verification of Ceilometer Operation

The Ceilometer services are named slightly differently on the Ubuntu and RHEL Server 7.0.

On Ubuntu, the service names are:

Polling agent	<code>ceilometer-agent-central</code> and <code>ceilometer-agent-compute</code>
Notification agent	<code>ceilometer-agent-notification</code>
Collector	<code>ceilometer-collector</code>

API Server	ceilometer-api
Alarms	ceilometer-alarm-evaluator and ceilometer-alarm-notifier

On RHEL Server 7.0, the service names are:

Polling agent	openstack-ceilometer-central and openstack-ceilometer-compute
Notification agent	openstack-ceilometer-notification
Collector	openstack-ceilometer-collector
API server	openstack-ceilometer-api
Alarms	openstack-ceilometer-alarm-evaluator and openstack-ceilometer-alarm-notifier

To verify the Ceilometer installation, users can verify that the Ceilometer services are up and running by using the `openstack-status` command.

For example, using the **openstack-status** command on an all-in-one node running Ubuntu 14.04.1 LTS with release 2.2 of Contrail installed shows the following Ceilometer services as active:

```

== Ceilometer services ==
ceilometer-api:           active
ceilometer-agent-central: active
ceilometer-agent-compute: active
ceilometer-collector:     active
ceilometer-alarm-notifier: active
ceilometer-alarm-evaluator: active
ceilometer-agent-notification:active

```

You can issue the `ceilometer meter-list` command on the OpenStack controller node to verify that meters are being collected, stored, and reported via the REST API. The following is an example of the output:

```

user@host:~# (source /etc/contrail/openstackrc; ceilometer meter-list)
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| Name                | Type    | Unit   | Resource ID                |
| User ID             | Project ID                |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| ip.floating.receive.bytes | cumulative | B      | a726f93a-65fa-4cad-828b-54dbfcf4a119 |
| None                 | None      |

```

ip.floating.receive.packets	cumulative	packet	a726f93a-65fa-4cad-828b-54dbfcf4a119	
None	None			
ip.floating.transmit.bytes	cumulative	B	a726f93a-65fa-4cad-828b-54dbfcf4a119	
None	None			
ip.floating.transmit.packets	cumulative	packet	a726f93a-65fa-4cad-828b-54dbfcf4a119	
None	None			
network	gauge	network	7fa6796b-756e-4320-9e73-87d4c52ecc83	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
network	gauge	network	9408e287-d3e7-41e2-89f0-5c691c9ca450	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
network	gauge	network	b3b72b98-f61e-4e1f-9a9b-84f4f3ddec0b	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
network	gauge	network	cb829abd-e6a3-42e9-a82f-0742db55d329	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
network.create	delta	network	7fa6796b-756e-4320-9e73-87d4c52ecc83	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
network.create	delta	network	9408e287-d3e7-41e2-89f0-5c691c9ca450	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
network.create	delta	network	b3b72b98-f61e-4e1f-9a9b-84f4f3ddec0b	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
network.create	delta	network	cb829abd-e6a3-42e9-a82f-0742db55d329	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
port	gauge	port	0d401d96-c2bf-4672-abf2-880eecf25ceb	
01edcedd989f43b3a2d6121d424b254d	82ab961f88994e168217ddd746fdd826			
port	gauge	port	211b94a4-581d-45d0-8710-c6c69df15709	
01edcedd989f43b3a2d6121d424b254d	82ab961f88994e168217ddd746fdd826			
port	gauge	port	2287ce25-4eef-4212-b77f-3cf590943d36	
01edcedd989f43b3a2d6121d424b254d	82ab961f88994e168217ddd746fdd826			
port.create	delta	port	f62f3732-222e-4c40-8783-5bcbc1fd6a1c	
01edcedd989f43b3a2d6121d424b254d	82ab961f88994e168217ddd746fdd826			
port.create	delta	port	f8c89218-3cad-48e2-8bd8-46c1bc33e752	
01edcedd989f43b3a2d6121d424b254d	82ab961f88994e168217ddd746fdd826			
port.update	delta	port	43ed422d-b073-489f-877f-515a3cc0b8c4	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
subnet	gauge	subnet	09105ed1-1654-4b5f-8c12-f0f2666fa304	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
subnet	gauge	subnet	4bf00aac-407c-4266-a048-6ff52721ad82	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
subnet.create	delta	subnet	09105ed1-1654-4b5f-8c12-f0f2666fa304	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			
subnet.create	delta	subnet	4bf00aac-407c-4266-a048-6ff52721ad82	
15c0240142084d16b3127d6f844adbd9	ded208991de34fe4bb7dd725097f1c7e			

```
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

NOTE: The `ceilometer meter-list` command lists the meters only if images have been created, or instances have been launched, or if subnet, port, floating IP addresses have been created, otherwise the meter list is empty. You also need to source the `/etc/contrail/openstackrc` file when executing the command.

Contrail Ceilometer Plugin

The Contrail Ceilometer plugin adds the capability to meter the traffic statistics of floating IP addresses in Ceilometer. The following meters for each floating IP resource are added by the plugin in Ceilometer.

```
ip.floating.receive.bytes
ip.floating.receive.packets
ip.floating.transmit.bytes
ip.floating.transmit.packets
```

The Contrail Ceilometer plugin configuration is done in the `/etc/ceilometer/pipeline.yaml` file when Contrail is installed by the Fabric provisioning scripts.

The following example shows the configuration that is added to the file:

```
sources:
  - name: contrail_source
    interval: 600
    meters:
      - "ip.floating.receive.packets"
      - "ip.floating.transmit.packets"
      - "ip.floating.receive.bytes"
      - "ip.floating.transmit.bytes"
    resources:
      - contrail://<IP-address-of-Contrail-Analytics-Node>:8081
    sinks:
      - contrail_sink
sinks:
  - name: contrail_sink
    publishers:
```

```
- rpc://
transformers:
```

The following example shows the Ceilometer meter list output for the floating IP meters:

```
+-----+-----+-----+
+-----+
+-----+-----+
+-----+-----+
| Name                | Type    | Unit    | Resource
ID                  |         |         |         | User ID
| Project ID          |         |         |         |
+-----+-----+-----+
+-----+
+-----+-----+
| ip.floating.receive.bytes | cumulative | B        | 451c93eb-
e728-4ba1-8665-6e7c7a8b49e2 | None
| None                |         |         |
| ip.floating.receive.bytes | cumulative | B        | 9cf76844-8f09-4518-a09e-
e2b8832bf894                | None
| None                |         |         |
| ip.floating.receive.packets | cumulative | packet   | 451c93eb-
e728-4ba1-8665-6e7c7a8b49e2 | None
| None                |         |         |
| ip.floating.receive.packets | cumulative | packet   | 9cf76844-8f09-4518-a09e-
e2b8832bf894                | None
| None                |         |         |
| ip.floating.transmit.bytes | cumulative | B        | 451c93eb-
e728-4ba1-8665-6e7c7a8b49e2 | None
| None                |         |         |
| ip.floating.transmit.bytes | cumulative | B        | 9cf76844-8f09-4518-a09e-
e2b8832bf894                | None
| None                |         |         |
| ip.floating.transmit.packets | cumulative | packet   | 451c93eb-
e728-4ba1-8665-6e7c7a8b49e2 | None
| None                |         |         |
| ip.floating.transmit.packets | cumulative | packet   | 9cf76844-8f09-4518-a09e-
e2b8832bf894                | None
| None                |         |         |
```

In the meter -list output, the Resource ID refers to the floating IP.

The following example shows the output from the `ceilometer resource-show -r 451c93eb-e728-4ba1-8665-6e7c7a8b49e2` command:

```
+-----+-----+
| Property | Value |
+-----+-----+
| metadata | {'router_id': u'None', u'status': u'ACTIVE', u'tenant_id': |
|          | u'ceed483222f9453ab1d7bcdd353971bc', u'floating_network_id': |
|          | u'6d0cca50-4be4-4b49-856a-6848133eb970', u'fixed_ip_address': |
|          | u'2.2.2.4', u'floating_ip_address': u'3.3.3.4', u'port_id': u'c6ce2abf- |
|          | ad98-4e56-ae65-ab7c62a67355', u'id': |
|          | u'451c93eb-e728-4ba1-8665-6e7c7a8b49e2', u'device_id': |
|          | u'00953f62-df11-4b05-97ca-30c3f6735ffd'} |
| project_id | None |
| resource_id | 451c93eb-e728-4ba1-8665-6e7c7a8b49e2 |
| source      | openstack |
| user_id     | None |
+-----+-----+
```

The following example shows the output from the `ceilometer statistics` command and the `ceilometer sample-list` command for the **ip.floating.receive.packets** meter:

```
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
+
| Period | Period Start          | Period End          | Count | Min | Max |
Sum      | Avg                  | Duration            | Duration Start      | Duration End          |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
+
| 0       | 2015-02-13T19:50:40.795000 | 2015-02-13T19:50:40.795000 | 2892 | 0.0 | 325.0 |
1066.0 | 0.368603042877 | 439069.674 | 2015-02-13T19:50:40.795000 | 2015-02-18T21:48:30.469000 |
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
+
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
| Resource ID          | Name                  | Type                | Volume |
Unit | Timestamp            |                      |         |
+-----+-----+-----+-----+-----+-----+
```

```

+-----+-----+
| 9cf76844-8f09-4518-a09e-e2b8832bf894 | ip.floating.receive.packets | cumulative | 208.0 |
packet | 2015-02-18T21:48:30.469000 |
| 451c93eb-e728-4ba1-8665-6e7c7a8b49e2 | ip.floating.receive.packets | cumulative | 325.0 |
packet | 2015-02-18T21:48:28.354000 |
| 9cf76844-8f09-4518-a09e-e2b8832bf894 | ip.floating.receive.packets | cumulative | 0.0 |
packet | 2015-02-18T21:38:30.350000 |

```

Ceilometer Installation and Provisioning

There are two scenarios possible for Contrail Ceilometer plugin installation.

1. If you install your own OpenStack distribution, you can install the Contrail Ceilometer plugin on the OpenStack controller node.
2. When using Contrail Cloud services, the Ceilometer controller services are installed and provisioned as part of the OpenStack controller node and the compute agent service is installed as part of the compute node when `enable_ceilometer` is set as `True` in the cluster **config** or **testbed** files.

Contrail OpenStack APIs

IN THIS CHAPTER

- [Working with Neutron | 122](#)

Working with Neutron

IN THIS SECTION

- [Data Structure | 122](#)
- [Network Sharing in Neutron | 123](#)
- [Commands for Neutron Network Sharing | 124](#)
- [Support for Neutron APIs | 124](#)
- [Contrail Neutron Plugin | 125](#)
- [DHCP Options | 125](#)
- [Incompatibilities | 126](#)

OpenStack's networking solution, Neutron, has representative elements for Contrail elements for Network (VirtualNetwork), Port (VirtualMachineInterface), Subnet (IpamSubnets), and Security-Group. The Neutron plugin translates the elements from one representation to another.

Data Structure

Although the actual data between Neutron and Contrail is similar, the listings of the elements differs significantly. In the Contrail API, the networking elements list is a summary, containing only the UUID, FQ name, and an href, however, in Neutron, all details of each resource are included in the list.

The Neutron plugin has an inefficient list retrieval operation, especially at scale, because it:

- reads a list of resources (for example. GET /virtual-networks), then

- iterates and reads in the details of the resource (GET /virtual-network/<uuid>).

As a result, the API server spends most of the time in this type of GET operation just waiting for results from the Cassandra database.

The following features in Contrail improve performance with Neutron:

- An optional detail query parameter is added in the GET of collections so that the API server returns details of all the resources in the list, instead of just a summary. This is accompanied by changes in the Contrail API library so that a caller gets returned a list of the objects.
- The existing Contrail list API takes in an optional parent_id query parameter to return information about the resource anchored by the parent.
- The Contrail API server reads objects from Cassandra in a multiget format into obj_uuid_cf, where object contents are stored, instead of reading in an xget/get format. This reduces the number of round-trips to and from the Cassandra database.

Network Sharing in Neutron

Using Neutron, a deployer can make a network accessible to other tenants or projects by using one of two attributes on a network:

- set the shared attribute to allow sharing
- set the router:external attribute, when the plugin supports an external_net extension

Using the Shared Attribute

When a network has the shared attribute set, users in other tenants or projects, including non-admin users, can access that network, using:

```
neutron net-list --shared
```

Users can also launch a virtual machine directly on that network, using:

```
nova boot <other-parameters> -nic net-id=<shared-net-id>
```

Using the Router:External Attribute

When a network has the router:external attribute set, users in other tenants or projects, including non-admin users, can use that network for allocating floating IPs, using:

```
neutron floatingip-create <router-external-net-id>
```

then associating the IP address pool with their instances.

NOTE: The VN hosting the FIP pool should be marked shared and external.

Commands for Neutron Network Sharing

The following table summarizes the most common Neutron commands used with Contrail.

Action	Command
List all shared networks.	<code>neutron net-list --shared</code>
Create a network that has the shared attribute.	<code>neutron net-create <net-name> -shared</code>
Set the shared attribute on an existing network.	<code>neutron net-update <net-name> -shared</code>
List all router:external networks.	<code>neutron net-list --router:external</code>
Create a network that has the router:external attribute.	<code>neutron net-create <net-name> -router:external</code>
Set the router:external attribute on an existing network.	<code>neutron net-update <net-name> -router:external</code>

Support for Neutron APIs

The OpenStack Neutron project provides virtual networking services among devices that are managed by the OpenStack compute service. Software developers create applications by using the OpenStack Networking API v2.0 (Neutron).

Contrail provides the following features to increase support for OpenStack Neutron:

- Create a port independently of a virtual machine.
- Support for more than one subnet on a virtual network.
- Support for allocation pools on a subnet.
- Per tenant quotas.
- Enabling DHCP on a subnet.
- External router can be used for floating IPs.

For more information about using OpenStack Networking API v2.0 (Neutron), refer to: <http://docs.openstack.org/api/openstack-network/2.0/content/> and the OpenStack Neutron Wiki at: <http://wiki.openstack.org/wiki/Neutron>.

Contrail Neutron Plugin

The Contrail Neutron plugin provides an implementation for the following core resources:

- Network
- Subnet
- Port

It also implements the following standard and upstreamed Neutron extensions:

- Security group
- Router IP and floating IP
- Per-tenant quota
- Allowed address pair

The following Contrail-specific extensions are implemented:

- Network IPAM
- Network policy
- VPC table and route table
- Floating IP pools

The plugin does not implement native bulk, pagination, or sort operations and relies on emulation provided by the Neutron common code.

DHCP Options

In Neutron commands, DHCP options can be configured using extra-dhcp-options in port-create.

Example

```
neutron port-create net1 --extra-dhcp-opt opt_name=<dhcp_option_name>,opt_value=<value>
```

The opt_name and opt_value pairs that can be used are maintained in GitHub: <https://github.com/Juniper/contrail-controller/wiki/Extra-DHCP-Options>.

Incompatibilities

In the Contrail architecture, the following are known incompatibilities with the Neutron API.

- Filtering based on any arbitrary key in the resource is not supported. The only supported filtering is by `id`, `name`, and `tenant_id`.
- To use a floating IP, it is not necessary to connect the public subnet and the private subnet to a Neutron router. Marking a public network with `router:external` is sufficient for a floating IP to be created and associated, and packet forwarding to it will work.
- The default values for quotas are sourced from `/etc/contrail/contrail-api.conf` and not from `/etc/neutron/neutron.conf`.

4

PART

Deploying Contrail Networking on the Cloud

[Multicloud Contrail Networking](#) | 128

Multicloud Contrail Networking

IN THIS CHAPTER

- [Deploying Contrail Multicloud with Contrail Command | 128](#)
- [Adding a Compute Host to Multicloud | 142](#)
- [Updating the Contrail Multicloud Cluster | 142](#)
- [Deleting the Contrail MultiCloud Cluster | 145](#)
- [Deploying Contrail Multicloud using REST API | 148](#)

Deploying Contrail Multicloud with Contrail Command

IN THIS SECTION

- [Deploying Microsoft Azure with Contrail Command | 130](#)
- [Deploying Amazon Web Services with Contrail Command | 135](#)
- [Deploying Google Cloud Platform \(GCP\) with Contrail Command | 137](#)

NOTE: The **Infrastructure: Multicloud** tab was removed from Contrail Command in Contrail Networking Releases 1912.L1 and 2008. The **Infrastructure: Multicloud** tab remains available in all other Contrail Networking Release 19 releases and Contrail Networking Release 20 releases through Release 2005.

All functionality provided within the **Infrastructure: Multicloud** tab in Contrail Command is available for evaluation purposes only. This functionality is not intended for deployment in production networks.

You can provision Contrail Multicloud with the Contrail Command UI.

Contrail supports provisioning of *Microsoft Azure*, *Amazon Web Services (AWS)*, and *Google Cloud Platform (GCP)*.

Multicloud gateway (*MC-GW*) node interconnects different Virtual Private Cloud (VPC)/Virtual Networks (VNETs) in cloud. Additionally, *MC-GW* extends on-premise resources to cloud.

This topic provides steps to configure *Microsoft Azure*, *Amazon Web Services (AWS)*, and *Google Cloud Platform (GCP)* with the Contrail Command UI.

Prerequisites:

- Contrail Multicloud is currently supported for deployments using Kubernetes as the orchestration platform only. See *Installing Standalone Kubernetes Contrail Cluster using the Contrail Command UI*.
- All the on-premise nodes except the *management* IPs must have private IPs.
- It is recommended to add static routes on all the control nodes towards the private subnets on the cloud.
- Control nodes and compute nodes must be on the same subnet and *MC-GW* must be on a different subnet.
- *MC-GW* node must be provisioned on RHEL 7.7.
- Compute nodes on the public cloud must be provisioned on RHEL 7.7.
- Instance type:
 - Azure—Standard_F2.
 - AWS—c4.xlarge or t2.xlarge.
 - Google Cloud Platform(GCP)—n1-standard-2
- Red Hat subscription with enabled packages must be available for on-premise MC-GW nodes.
- Time must be synchronized on all the nodes with NTP.
- *contrail-command* node must have connectivity to ToRs.
- You must enable NETCONF on ToRs connected to the on-premise MC-GW nodes.
- You must configure IPTABLES on the on-premise MC-GW nodes with *INPUT* and *FORWARD* and default *ACCEPT* policy.
- For Azure deployment, you must have *subscription* and *resource group*.

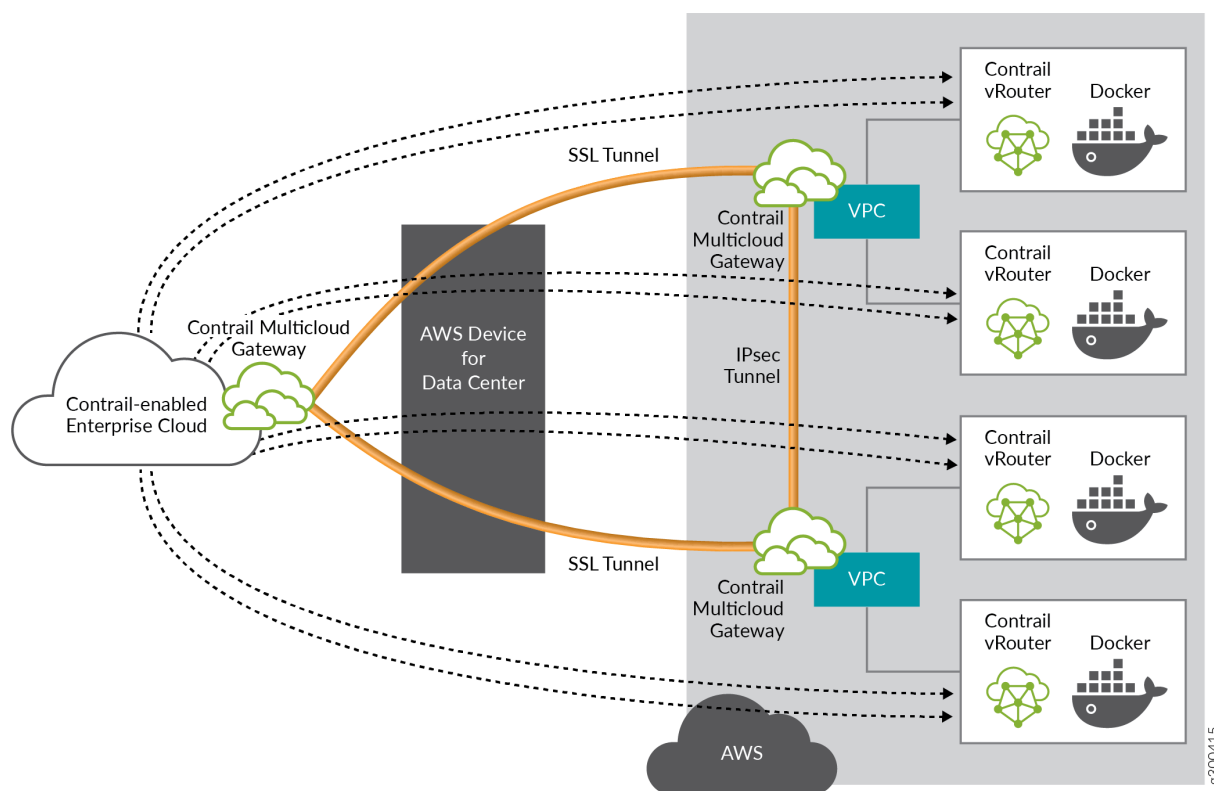
For details, refer to [Creating a Resource Group](#).

- For provisioning Microsoft Azure with Contrail Command, you must have Azure account. For details, refer to <https://docs.microsoft.com/en-us/learn/modules/create-an-azure-account/>.

For provisioning Amazon Web Services (AWS) with Contrail Command, you must have AWS account. For details, refer to <https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/>.

- For provisioning Google Cloud Platform (GCP) with Contrail Command, you must have a GCP account. For details, see [Creating and managing service accounts](#) within the [Cloud Identity and Access Management documentation](#) for GCP.

Sample Topology:



Deploying Microsoft Azure with Contrail Command

To provision *Microsoft Azure*:

1. Login to the desired cluster from the Contrail Command UI.
 - Select the desired cluster from the **Selected Cluster** drop down list.
 - Enter the **Username** and **Password** for the cluster.
2. Click **Multi Cloud**.
3. Click **Add**.

- a. Select *Azure* from the drop down list of **Type of Cloud**.
Enter **Cloud Name** and **Organization Name**.

Click **Expand All**

The screenshot shows the 'Create Cloud' interface in the Contrail Command application. The left sidebar has 'Multi Cloud' selected. The main form area is titled 'Create Cloud' and contains the following fields:

- Type Of Cloud:** A dropdown menu with 'Azure' selected.
- Cloud Name:** A text input field containing 'srecloud'.
- Organization Name:** A text input field containing 'Juniper'.
- Keypair Name:** An empty text input field.
- SSH Key Directory path:** An empty text input field.
- Expand All / Collapse All:** Two buttons, with 'Expand All' being active.
- Region Details:** A section that is expanded, containing:
 - Region Name*:** A dropdown menu with 'WestUS' selected.
 - Resource Group:** A text input field containing 'scaleTest'.

- b. Enter the required details including **Cloud Name**, **Region Details**, **VNET Details**, **Security Groups**, **Instances**.

Enter the **Resource Group** which was created earlier by following [Creating a Resource Group](#) procedure.

CONTRAIL COMMAND

INFRASTRUCTURE

Multi Cloud

Create Cloud

default-domain

default-project

Admin

Servers

Cluster

Fabrics

Multi Cloud

Networks

Expand All Collapse All

Region Details

Region Name*

WestUS

Resource Group

scaleTest

VNET Details

Name

vnet1

cidr_block

172.16.1.0/24

Private Subnet CIDR

CIDR block*

172.16.1.0/25

Availability zone

1

Security Groups

Rule Name

AzureCloudIn

Direction

Ingress

From port

To Port

CIDR Block

0.0.0.0/0

Protocol

ANY

Rule Name

AzureCloudout

Direction

Egress

From port

To Port

CIDR Block

0.0.0.0/0

Protocol

ANY

+ Add

Add **Subnet** for Compute node and Controller node.

Add **Subnet** for on-premise gateway nodes.

Keypair Name and **SSH Key Directory Path** are not required for Azure deployment. Azure generates these values in the back-end.

- c. Click **Create**.

You can access the logs at `logs/var/log/contrail/cloud.log` on the Contrail Command server.

4. Click **Multi Cloud**.

You must see your multi clouds listed here with the **Status** as color *Green*.

STATUS	NAME	ORGANIZATION	TYPE
●	snccloud	Juniper	Public

5. Click **Servers**.

- a. Click **Create**.

- i. Enter the required details for the on-premise gateway nodes.

ii. Click **Create**.

You can access the logs at `logs/var/log/contrail/cloud.log`.

6. Click **Cluster**.

a. Click **Subcluster**.

b. Click **Add Subcluster**.

c. Click **Add Existing VPC**.

i. Add the required details.

ii. Select the created *Azure* cloud from the drop down list of **Select Existing Cloud**.

- iii. The **Public MultiCloud GW Role** must be the name of the earlier created *Azure* GW.

From the drop down list, select **User Credentials** of the on-premise private cloud.

- iv. Check the *deployment logs* at `/var/log/contrail/cloud.log` and `/var/log/contrail/deploy.log` on the Contrail Command server.
- v. Click **Create**.

You can access the logs at `logs/var/log/contrail/cloud.log` and `logs/var/log/contrail/deploy.log` on the Contrail Command server.

Deploying Amazon Web Services with Contrail Command

To provision *Amazon Web Services (AWS)*:

1. Login to the desired cluster from the Contrail Command UI.
 - Select the desired cluster from the **Selected Cluster** drop down list.
 - Enter the **Username** and **Password** for the cluster.
2. Click **Multi Cloud**.
3. Click **Add**.
 - a. Select *AWS* from the drop down list of **Type of Cloud**.
 - b. Enter the required details including **SSH User**, **Cloud Name**, **AWS Credentials**, **Region Details**, **VPC Details**, **Security Groups**, **Instances**.

The screenshot shows the 'Multi-Cloud' configuration page in the Contrail Command interface. The left sidebar has 'Multi-Cloud' selected. The main area contains several sections:

- Basic Info:** Fields for 'Cloud Name' (set to 'snccloud'), 'Region Name' (set to 'us-east-1'), and 'Billing Provider' (set to 'AWS').
- Add Credentials:** Fields for 'Access Key' and 'Secret Key'.
- VPC Details:** Fields for 'VPC Name' (set to 'snc-vpc'), 'VPC ID' (set to 'vpc-1a1b2c3d'), and 'CIDR Block' (set to '10.0.0.0/16').
- Security Groups:** Two rows for 'Rule Name', 'Direction', 'From Port', 'To Port', 'CIDR Block', and 'Protocol'. The first row is for 'snc-sg-1' and the second for 'snc-sg-2'.
- Subnets:** Four rows for 'Subnet Name', 'VPC Name', 'Subnet ID', 'CIDR Block', 'Availability Zone', 'Subnet Type', and 'Subnet Role'. The subnets are 'snc-sg-1', 'snc-sg-2', 'snc-sg-3', and 'snc-sg-4'.

At the bottom, there are 'Create' and 'Cancel' buttons.

c. Click **Create**.

4. Assign private Multicloud Gateway nodes.

5. Add Gateways BGP Peer.

6. Click **Multi Cloud**.

You must see your multi clouds listed here with the **Status** as color *Green*.

The screenshot shows the 'Multi-Cloud' table in the Contrail Command interface. The table has columns for 'STATUS', 'NAME', 'ORGANIZATION', and 'TYPE'. There is one entry with a green status icon.

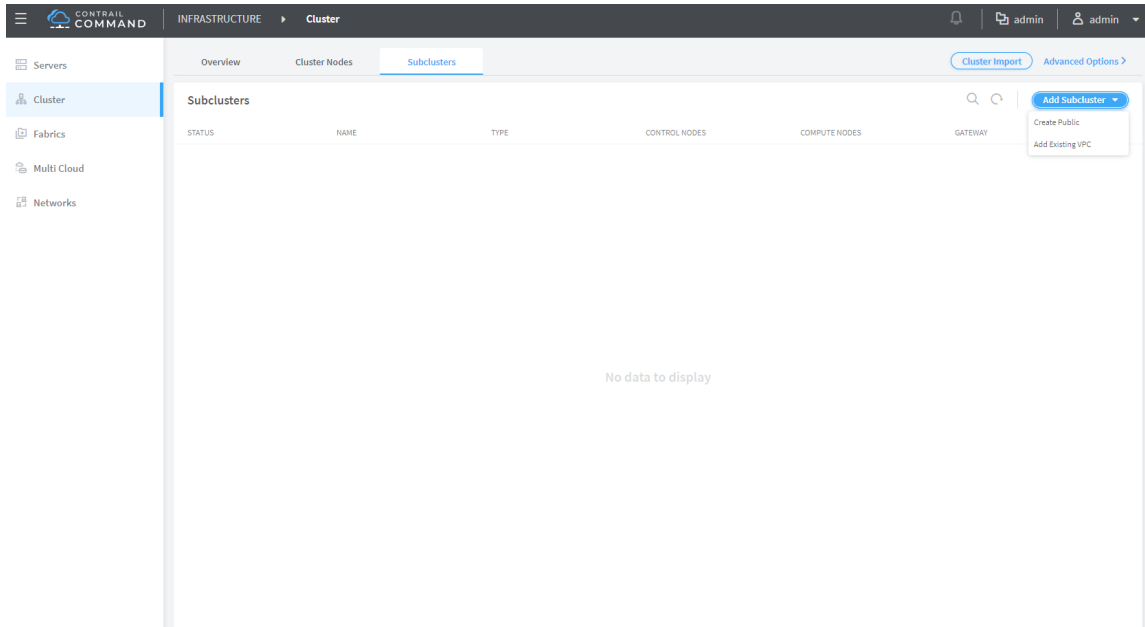
STATUS	NAME	ORGANIZATION	TYPE
●	snccloud	Juniper	Public

7. Click **Cluster**.

a. Click **Subcluster**.

b. Click **Add Subcluster**.

c. Click **Add Existing VPC**.



- i. Add the required details.

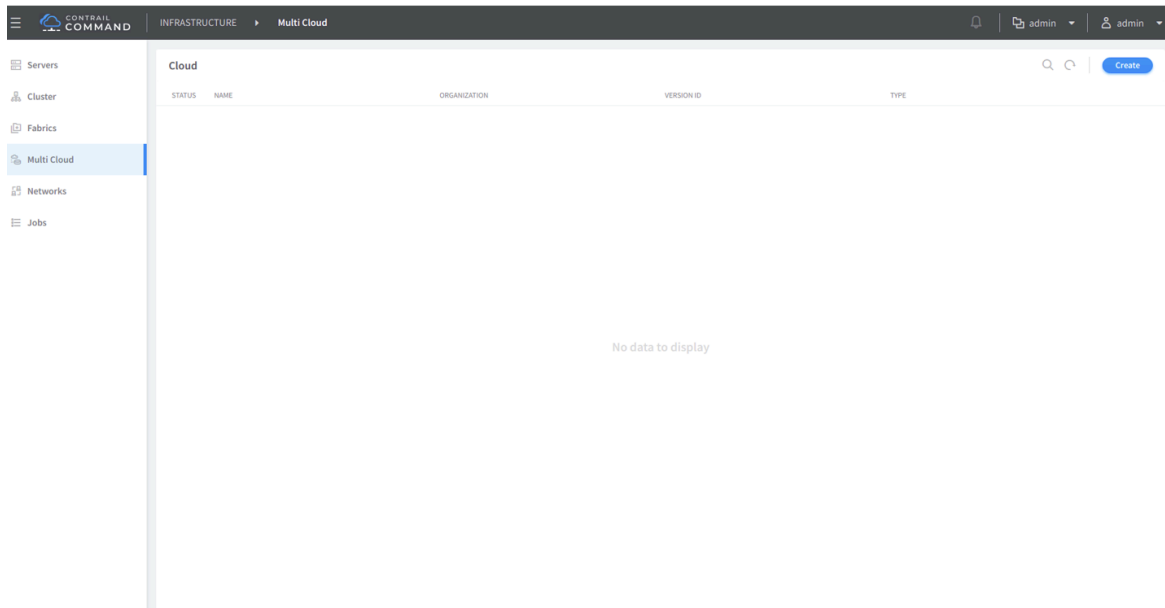
- ii. Select the created *AWS* cloud from the drop down list of **Select Existing Cloud**
- iii. The **Public MultiCloud GW Role** must be the name of the earlier created *AWS* GW.
- iv. Click **Create**.

Deploying Google Cloud Platform (GCP) with Contrail Command

Starting with Contrail Networking Release 1911, you can provision Google Cloud Platform (GCP) cloud networks within Contrail Command.

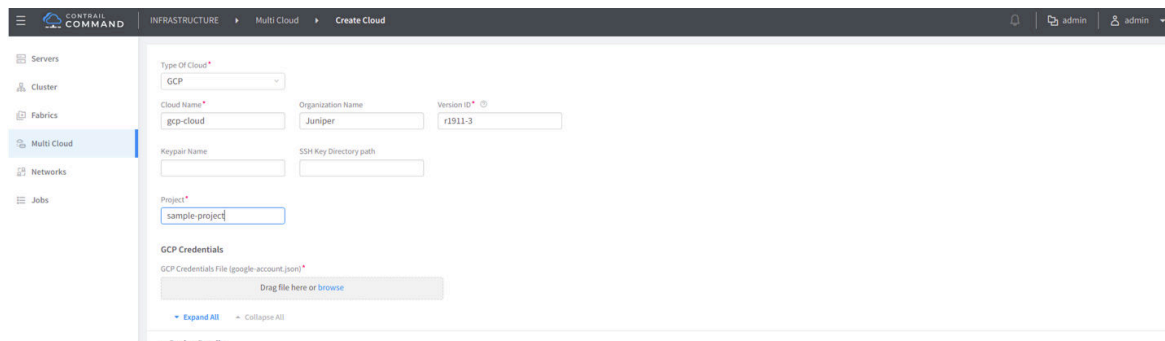
To provision Google Cloud Platform (GCP):

1. Login to the desired cluster from the Contrail Command UI.
 - Select the desired cluster from the **Selected Cluster** drop down list.
 - Enter the **Username** and **Password** for the cluster.
2. Click **Multi Cloud**.
3. Click **Create**.



4. Select **GCP** from the **Type of Cloud** dropdown list.
5. Enter a **Cloud Name**, **Organization Name**, **Version ID**, and **Project**:
6. Upload the GCP credentials file (*google-account.json*).

GCP credentials files are created from Google Cloud. See [Creating and managing service account keys](#) in the [Cloud Identity and Access Management documentation](#) for the Google Cloud Platform.



7. Enter required **Region Details** and **VPC Details**.

Region Details

Region Name*
us-central1

VPC Details

Name* CIDR block*
vpc-gcp-test 172.16.81.149/2

Private Subnet CIDR

Private Subnet Name* CIDR block* Availability zone
subnet-test 172.16.81.149/2 a

+ Add

8. Enter required **Firewall Rules**, and **Instances**. One instance must include the **Gateway** role.

Firewall Rules

Rule Name	Direction	CIDR Block	Protocol	List of ports to allow
rule-egr	Egress	0.0.0.0/0	ANY	+ Add
rule-ing	Ingress	0.0.0.0/0	ANY	+ Add

+ Add

Instances

Name*	Role*	Volume	Subnet*	Instance Type*
gw-node	Gateway	20	172.16.81.149/2	n1-standard-2

Operating System*
RedHat Enterprise Lin...

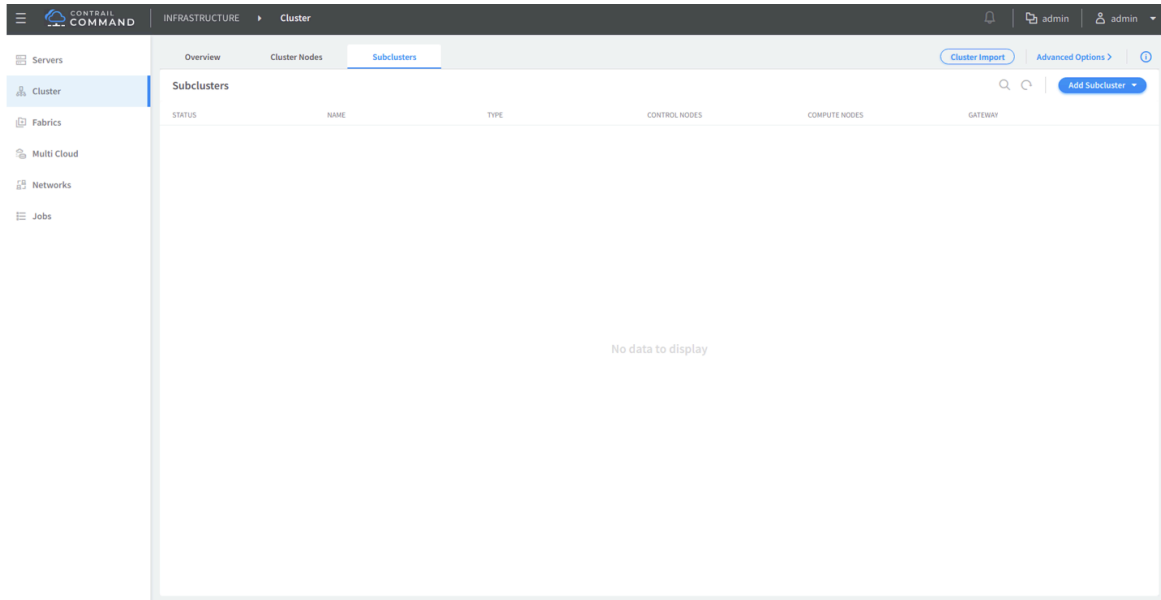
+ Add

9. Click **Create**.
10. You are returned to the main **Multi Cloud** page after the GCP instance is created. Click **Multi Cloud** if you are not moved to this page.

Confirm that your GCP instance is created and that the **Status** is *Green*.

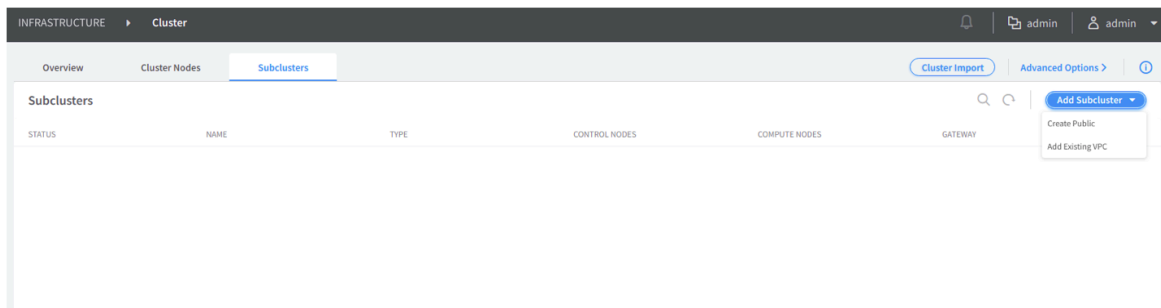
INFRASTRUCTURE ▸ Multi Cloud					admin	
Cloud					Create	
STATUS	NAME	ORGANIZATION	VERSION ID	TYPE		
●	gcp-cloud	Juniper	image-test	Public	...	

11. Click **Cluster**.
12. Click **Subcluster**.



13. Click **Add Subclusters**.

14. Click **Add Existing VPC**.



15. Add the required details.

16. Select the created GCP cloud from the **Select Existing Cloud** drop-down list.

17. Upload the GCP credentials file (*google-account.json*).

GCP credentials files are created from Google Cloud. See [Creating and managing service account keys](#) in the [Cloud Identity and Access Management documentation](#) for the Google Cloud Platform.

Name *

gcp-test

Select Existing Cloud *

gcp-cloud

Enter AWS Credentials

Access Key

Secret Key

Or Upload Credentials File (*.csv)

Drag file here or [browse](#)

Azure Credentials

Client ID

Tenant ID

Subscription ID

Client secret

Or Upload Credentials File (*.csv)

Drag file here or [browse](#)

GCP Credentials

GCP Credentials File (google-account.json)

Drag file here or [browse](#)

google-account.json

User Credentials *

gcp-cloud-credential-6...

18. The Public MultiCloud GW Role must be the name of the earlier created GCP GW.

Public MultiCloud GW Role

Public MultiCloud Gateway

gwnode

VPN

ipsec_client

+ Add

Private MultiCloud GW Role

Private MultiCloud Gateway *

test

Gateway *

172.16.81.149

VPN

ipsec_client

Gateway Services

bgp_rr

+ Add

BFD multiplier

5

VPN network

192.64.0.0/16

BFD interval

200ms

BFD interval multihop

500ms

OnPrem Private Subnets

172.16.81.149

+ Add

BFD multiplier multihop

5

AS

65000

OpenVPN port number

443

VPN loopback network

192.65.0.0/16

Release History Table

Release	Description
1911	Starting with Contrail Networking Release 1911, you can provision Google Cloud Platform (GCP) cloud networks within Contrail Command.

RELATED DOCUMENTATION

Installing Contrail Command

Importing Contrail Cluster Data using Contrail Command

Adding a Compute Host to Multicloud

You can modify the multicloud topology by adding a new compute host to the VPC as well as a new VPC altogether.

1. Edit the **topology.yml** to reflect your new topology. Ensure that the new VPC uses a different IP address pool.

```
# vi topology.yml
```

2. Navigate to the **one-click-deployer** directory.

```
# cd multicloud/one-click-deployer
```

3. Run the **modify.sh** script to generate the topology and deploy Contrail.

```
# ./modify.sh
```

NOTE: You can download the scripts from <https://ssd-git.juniper.net/contrail/contrail-multi-cloud/tree/master/one-click-deployer>. If you are not able to access the page, you might not have the required access permission. E-mail eng-git-support@juniper.net for necessary permissions.

Updating the Contrail Multicloud Cluster

IN THIS SECTION

- [Adding Another Gateway to an Existing VPC | 143](#)

Perform the following procedure to update the Contrail multicloud cluster with adding another gateway to an existing Virtual Private Cloud (VPC).

Adding Another Gateway to an Existing VPC

To add another gateway to an existing Virtual Private Cloud (VPC):

1. Run the following request payload to create the gateway node object.

The UUIDs listed in `cloud_security_group_refs`, `tag_refs`, and `cloud_private_subnet_refs` were created in the topic “Deploy Public Cloud Infrastructure” in ["Deploying Contrail Multicloud using REST API" on page 148](#).

Example: `create_new_public_gw_node.yml`

```
resources:
- data:
    uuid: 39845468-903b-4d69-b6d3-dec647ec223e
    name: public_gateway_node_B
    parent_type: global-system-config
    fq_name:
    - default-global-system-config
    - public_gateway_node_B
    perms2:
      owner: admin
    hostname: gateway_B
    interface_name: eth1
    type: private
    cloud_info:
      availability_zone: a
      machine_id: ami-18726478
      instance_type: t2.xlarge
      roles:
      - gateway
    cloud_private_subnet_refs:
    - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaeac5798d
    credential_refs:
    - uuid: 9d0fffff-3fd8-439c-bdb2-ff5800497579
    cloud_security_group_refs:
    - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaeac57123
    tag_refs:
    - uuid: efd769a8-2e6c-11e9-b210-d663bd873d93
    kind: node
    operation: CREATE
```

2. Create the contrail-multicloud-gw-node object.

The UUID listed in `node_refs` was created in Step "1" on page 143.

Example: `create_new_mcgw_node.yml`

```
resources:
- data:
    name: public_contrail_multicloud_gw_node_B
    node_refs:
    - uuid: 39845468-903b-4d69-b6d3-dec647ec223e
    protocols_mode:
    - ssl_server
    - ipsec_server
    - ipsec_client
    - ssl_client
    parent_type: contrail-cluster
    parent_uuid: a5063dde-2681-11e9-8021-0050568a3bf0
    kind: contrail_multicloud_gw_node
    operation: CREATE
```

3. Update the cloud object with `provisioning_state` as `NOSTATE`. This cloud object was created in the topic "Deploy Public Cloud Infrastructure" in "Deploying Contrail Multicloud using REST API" on page 148.

```
resources:
- data:
    provisioning_state: NOSTATE
    uuid: dfb40e0d-c9f4-47cd-bd5c-1efdd28fd4fc
    kind: cloud
    Operation: UPDATE
```

4. Update cluster object with `provisioning_state` as `NOSTATE` and `provisioning_action` as `UPDATE_CLOUD`.

```
resources:
- data:
    uuid: a5063dde-2681-11e9-8021-0050568a3bf0
    provisioning_state: NOSTATE
    provisioning_action: UPDATE_CLOUD
```

```
kind: contrail_cluster
operation: UPDATE
```

RELATED DOCUMENTATION

[Deploying Contrail Multicloud using REST API | 148](#)

[Deleting the Contrail MultiCloud Cluster | 145](#)

Deleting the Contrail MultiCloud Cluster

IN THIS SECTION

- [Removing Extension of Public Cloud to On-Premise Contrail Cluster | 145](#)
- [Deleting Public Cloud Infrastructure Objects | 147](#)

Perform the following procedures to delete the Contrail multicloud cluster:

- Remove the extension of the public cloud to the on-premise Contrail cluster.
- Delete the public cloud infrastructure objects.

Removing Extension of Public Cloud to On-Premise Contrail Cluster

To remove the extension of public cloud to the on-premise Contrail cluster:

1. The following request uses DELETE_CLOUD in provisioning_action to remove the extension.

Example: delete_cloud.yml

```
---
resources:
- data:
    uuid: a5063dde-2681-11e9-8021-0050568a3bf0
    provisioning_state: NOSTATE
    provisioning_action: DELETE_CLOUD
```



```

cloud_refs: []
mc_gw_info: {}
kind: contrail_cluster
operation: UPDATE

```

2. In addition, delete the on-premise cloud objects that were created earlier. If the UUID of the resource is unknown, use **contrailcli** to get them.

Example: delete_onprem_cloud_objects.yml

```

---
resources:
- data:
    uuid: 0f11f71c-f451-11e8-bccc-a4d18newdcd4
    operation: DELETE
    kind: contrail_multicloud_gw_node

- data:
    cloud_private_subnet_refs: []
    tag_refs: []
    uuid: 41f99f2d-a5a4-4e2c-b598-c173cf748953
    kind: node
    operation: UPDATE

- data:
    cloud_private_subnet_refs: []
    tag_refs: []
    uuid: c8d9d4ec-2f4a-11e9-bfac-0050568a3bf0
    kind: node
    operation: UPDATE

- data:
    cloud_private_subnet_refs: []
    tag_refs: []
    uuid: c8d9c1b4-2f4a-11e9-bfac-0050568a3bf0
    kind: node
    operation: UPDATE

- data:
    uuid: fdc97d65-9f58-4b5c-ac8c-07341a115ab5
    kind: cloud_private_subnet

```

```

operation: DELETE

- data:
  uuid: fdc97d65-9f58-4b5c-ac8c-07341c334ab8
  kind: cloud_private_subnet
  operation: DELETE

- data:
  uuid: 4bd887b1-3f65-59c1-bc2f-ZGZiYVWhY2Vmejc2NTQK
  kind: virtual_cloud
  operation: DELETE

- data:
  uuid: 70d4d63e-6f51-4181-bb0b-4a1bbf242332
  kind: cloud_region
  operation: DELETE

- data:
  uuid: 061a024a-9da0-40a3-974b-9309dfd85255
  kind: cloud_provider
  operation: DELETE

- data:
  uuid: dfb40e0d-c9f4-47cd-bd5c-MWVmZGQyOGZkNGZjCg
  kind: cloud
  operation: DELETE

- data:
  uuid: 4e77005b-b7ba-489b-9891-aGFjawo9eadf
  kind: cloud_user
  operation: DELETE

- data:
  uuid: 2eefeb06-2e7c-11e9-b210-d663bd873d93
  kind: tag
  operation: DELETE

```

Deleting Public Cloud Infrastructure Objects

Use the following request to delete public cloud infrastructure objects:

Example: delete_public_cloud_objects.yml

```
---
resources:
- data:
    provisioning_state: NOSTATE
    uuid: dfb40e0d-c9f4-47cd-bd5c-1efdd28fd4fc
    provisioning_action: DELETE_CLOUD
  kind: cloud
  operation: UPDATE
```

RELATED DOCUMENTATION

[Deploying Contrail Multicloud using REST API | 148](#)

[Updating the Contrail Multicloud Cluster | 142](#)

Deploying Contrail Multicloud using REST API

IN THIS SECTION

- [Prerequisites and Assumptions | 148](#)
- [Objective and Workflow | 149](#)
- [Deploying the Public Cloud Infrastructure | 149](#)
- [Creating Contrail Roles Specific to Public Cloud Instances | 155](#)
- [Creating On-Premise Cloud Objects | 157](#)
- [Extending On-Premise Contrail Cluster to Public Cloud | 164](#)

This section explains how to deploy Contrail Multicloud using REST API.

Prerequisites and Assumptions

The following are the assumptions for Contrail Multicloud deployment:

- Contrail Controller cluster is already deployed on the on-premise side. On-premise implies that the software is installed locally on the organization or data center servers.
- All nodes on the on-premise side have a management IP (declared in node object as `ip_address` field) control/data IP as a child port object `ip_address` field.
- Ensure that static routes are added in the on-premise multicloud gateway, Contrail controller, and top-of-rack (TOR) node.
- Ensure that the on-premise Contrail cluster and multicloud gateway are in two different subnets.
- `contrail-api` is listening on the control-data IP.
- Understand the `contrail-go-api` server tools and concepts.
 - <https://github.com/Juniper/contrail/blob/master/doc/index.md>
 - <https://github.com/Juniper/contrail/blob/master/doc/cli.md>
 - https://github.com/Juniper/contrail/blob/master/doc/rest_api.md
 - https://github.com/Juniper/contrail/blob/master/doc/rest_api.md#sync-api

Objective and Workflow

The deployment consists of the following steps:

1. Create an entire public cloud infrastructure that includes Virtual Private Cloud (VPC)/virtual network, virtual machines, routes, and so on.
2. Deploy multicloud gateway roles for both on-premise site and public cloud sites.
3. Deploy Contrail and Kubernetes components needed on the public cloud site.
4. Establish connectivity between on-premise site and public cloud.

Deploying the Public Cloud Infrastructure

When deploying the following example `deploy_public_cloud_infra.yml` file, multiple resources for Amazon Web Services (AWS) infrastructure are created. Summarized are important resources created using this yaml file.

- One VPC (192.168.100.0/24)
- One private subnet (192.168.100.128/25).
- Two security group rules.

- Two Elastic Compute Cloud (EC2) instances (gateway and compute)

Verify that the correct access key and secret key are entered in the `cloud_user` object.

Example: `deploy_public_cloud_infra.yml`

```
---
resources:
- data:
    name: public_cloud_tag
    uuid: efd769a8-2e6c-11e9-b210-d663bd873d93
    fq_name:
    - public_cloud_tag
    tag_type_name: label
    tag_value: public_cloud_provider_aws
  kind: tag
  operation: CREATE
- data:
    name: public_cloud_key
    uuid: 4e77005b-b7ba-489b-9891-8472cee8ghts
    parent_type: global-system-config
    fq_name:
    - default-global-system-config
    - public_cloud_key
  kind: keypair
  operation: CREATE
- data:
    name: public_cloud_credential
    uuid: 9d0fffff-3fd8-439c-bdb2-ff5800497579
    parent_type: global-system-config
    fq_name:
    - default-global-system-config
    - public_cloud_credential
    ssh_user: ec2-user
    keypair_refs:
    - uuid: 4e77005b-b7ba-489b-9891-8472cee8ghts
  kind: credential
  operation: CREATE
- data:
    uuid: 4e77005b-b7ba-489b-9891-8472cee9eadf
    name: public_cloud_user
    fq_name:
```

```

- public_cloud_user
perms2:
  owner: admin
aws_credential:
  access_key: xxxxxxxxx
  secret_key: YYYYYYYYYYYYYYYYYYYY
credential_refs:
- uuid: 9d0fffff-3fd8-439c-bdb2-ff5800497579
kind: cloud_user
operation: CREATE
- data:
  provisioning_state: CREATED
  uuid: dfb40e0d-c9f4-47cd-bd5c-1efdd28fd4fc
  name: public_cloud
  fq_name:
  - public_cloud
  perms2:
    owner: admin
  organization: test
  project: 5.0.3
  cloud_user_refs:
  - uuid: 4e77005b-b7ba-489b-9891-8472cee9eadf
kind: cloud
operation: CREATE
- data:
  name: public_cloud_provider
  parent_type: cloud
  fq_name:
  - public_cloud
  - public_cloud_provider
  perms2:
    owner: admin
  type: aws
kind: cloud_provider
operation: CREATE
- data:
  name: us-west-1
  parent_type: cloud-provider
  fq_name:
  - public_cloud
  - public_cloud_provider
  - public_cloud_region
  display_name: public_cloud_region

```

```

    perms2:
      owner: admin
kind: cloud_region
operation: CREATE
- data:
  name: publc_virtual_cloud
  parent_type: cloud-region
  fq_name:
    - public_cloud
    - public_cloud_provider
    - public_cloud_region
    - publc_virtual_cloud
  perms2:
    owner: admin
  cidr_block: 192.168.100.0/24
  tag_refs:
    - uuid: efd769a8-2e6c-11e9-b210-d663bd873d93
kind: virtual_cloud
operation: CREATE
- data:
  uuid: 4bd887b1-3f65-59c1-bc2f-dfbaeac5798d
  name: public_cloud_private_subnet
  parent_type: virtual-cloud
  fq_name:
    - public_cloud
    - public_cloud_provider
    - public_cloud_region
    - publc_virtual_cloud
    - public_cloud_private_subnet
  perms2:
    owner: admin
  cidr_block: 192.168.100.128/25
  availability_zone: a
kind: cloud_private_subnet
operation: CREATE
- data:
  uuid: 4bd887b1-3f65-59c1-bc2f-dfbaeac57123
  name: public_cloud_security_group
  parent_type: virtual-cloud
  fq_name:
    - public_cloud
    - public_cloud_provider
    - public_cloud_region

```

```

- public_virtual_cloud
- public_cloud_security_group
perms2:
  owner: admin
kind: cloud_security_group
operation: CREATE
- data:
  name: public_cloud_security_group_rule_ingress
  parent_type: cloud-security-group
  fq_name:
    - public_cloud
    - public_cloud_provider
    - public_cloud_region
    - public_virtual_cloud
    - public_cloud_security_group
    - public_cloud_security_group_rule_ingress
  perms2:
    owner: admin
  direction: ingress
  protocol: "-1"
  from_port: 0
  to_port: 0
  cidr_block: 0.0.0.0/0
kind: cloud_security_group_rule
operation: CREATE
- data:
  name: public_cloud_security_group_rule_egress
  parent_type: cloud-security-group
  fq_name:
    - public_cloud
    - public_cloud_provider
    - public_cloud_region
    - public_virtual_cloud
    - public_cloud_security_group
    - public_cloud_security_group_rule_egress
  perms2:
    owner: admin
  direction: egress
  protocol: "-1"
  from_port: 0
  to_port: 0
  cidr_block: 0.0.0.0/0
kind: cloud_security_group_rule

```



```

    operation: CREATE
  - data:
      uuid: 4bd887b1-3f65-59c1-bc2f-dfbaenew43526
      name: public_gateway_node
      parent_type: global-system-config
      fq_name:
        - default-global-system-config
        - public_gateway_node
      perms2:
        owner: admin
      hostname: gateway
      interface_name: eth1
      type: private
      cloud_info:
        availability_zone: a
        machine_id: ami-18726478
        instance_type: t2.xlarge
        roles:
          - gateway
      cloud_private_subnet_refs:
        - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaeac5798d
      credential_refs:
        - uuid: 9d0fffff-3fd8-439c-bdb2-ff5800497579
      cloud_security_group_refs:
        - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaeac57123
      tag_refs:
        - uuid: efd769a8-2e6c-11e9-b210-d663bd873d93
    kind: node
    operation: CREATE
  - data:
      uuid: 4bd887b1-3f65-59c1-bc2f-dfbaenew43634
      name: public_compute_node
      parent_type: global-system-config
      fq_name:
        - default-global-system-config
        - public_compute_node
      perms2:
        owner: admin
      hostname: compute
      interface_name: eth0
      type: private
      cloud_info:
        availability_zone: a

```

```

machine_id: ami-18726478
instance_type: t2.xlarge
volume_size: 24
roles:
  - compute
cloud_private_subnet_refs:
  - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaeac5798d
credential_refs:
  - uuid: 9d0fffff-3fd8-439c-bdb2-ff5800497579
cloud_security_group_refs:
  - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaeac57123
tag_refs:
  - uuid: efd769a8-2e6c-11e9-b210-d663bd873d93
kind: node
operation: CREATE

```

Creating Contrail Roles Specific to Public Cloud Instances

Use these guidelines in the `create_contrail_roles_for_cloud_objects.yml` file in this procedure:

- `a5063dde-2681-11e9-8021-0050568a3bf0` is the `contrail_cluster` UUID. Also `a50635c8-2681-11e9-8021-0050568a3bf0` is the `kubernetes_cluster` UUID. It is assumed that both UUIDs are already created.
- `node_refs` is the UUID of the nodes that were created in the previous topic “Deploy Public Cloud Infrastructure.”
- `parent_uuid` is the UUID of the `kubernetes_cluster` for the `kubernetes_node` role object.
- For `contrail_multicloud_gw_node` and `contrail_vrouter_node` role, object `parent_uuid` is the `contrail_cluster` objects UUID.

To create Contrail roles specific to public cloud instances, perform the following steps:

1. Enter these requests to locate the Contrail and Kubernetes cluster UUIDs.

```

contrailcli list contrail_cluster | grep uuid
contrailcli list kubernetes_cluster | grep uuid

```

2. Use the following request payload to create the contrail roles for cloud objects.

Example: create_contrail_roles_for_cloud_objects.yml

```
---
resources:
- data:
    name: public_contrail_multicloud_gw_node
    node_refs:
    - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaenew43526
    protocols_mode:
    - ssl_server
    - ipsec_server
    - ipsec_client
    parent_type: contrail-cluster
    parent_uuid: a5063dde-2681-11e9-8021-0050568a3bf0
    kind: contrail_multicloud_gw_node
    operation: CREATE

- data:
    name: public_kubernetes_node
    node_refs:
    - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaenew43634
    parent_type: kubernetes-cluster
    parent_uuid: a50635c8-2681-11e9-8021-0050568a3bf0
    kind: kubernetes_node
    operation: CREATE

- data:
    name: public_contrail_vrouter_node
    node_refs:
    - uuid: 4bd887b1-3f65-59c1-bc2f-dfbaenew43634
    parent_type: contrail-cluster
    parent_uuid: a5063dde-2681-11e9-8021-0050568a3bf0
    kind: contrail_vrouter_node
    operation: CREATE
```

3. Update the provisioning_state of cloud object to NOSTATE to trigger the deployment of the public cloud.

```
- data:
    provisioning_state: NOSTATE
```

```

    uuid: dfb40e0d-c9f4-47cd-bd5c-1efdd28fd4fc
    kind: cloud
    Operation: UPDATE

```

Wait for the cloud deployment logs in `/var/log/contrail/cloud.log` to complete before proceeding to the next steps. When completed, the `provisioning_state` of the cloud resource change from `NOSTATE` to either `UPDATED` or `UPDATE_FAILED`.

Creating On-Premise Cloud Objects

In the following `create_onprem_pvt_port.yml` file, node objects were already created. You are updating the `cloud_private_subnet` and `tag_refs`. Per the requirement, you need to have multicloud gateway and other roles on the on-premise cluster (Contrail controller, Kubernetes nodes, OpenStack nodes) on two different networks connected through a TOR. Hence, in the following yaml file there are two private subnets created. Be careful when adding the `cloud_private_subnet_refs` to the nodes.

To create on-premise cloud objects:

1. Create the private port.

If the private interface is not already created for the on-premise cluster nodes by using the UI, create them here. In the following example file, you are using the UUID of each on-premise cluster node resource.

Example: `create_onprem_pvt_port.yml`

```

---
resources:
# Create private interface for onprem compute node
- data:
    parent_type: node
    parent_uuid: 4bd887b1-3f65-59c1-bc2f-dfbaenew43634
    name: bond0
    ip_address: 192.168.1.2
    pxe_enabled: false
    kind: port

# Create private interface for onprem controller
- data:
    parent_type: node
    parent_uuid: c8d9d4ec-2f4a-11e9-bfac-0050568a3bf0
    name: bond0

```

```

    ip_address: 192.168.1.1
    pxe_enabled: false
  kind: port

```

2. Create onprem tag. This is used later to link on-premise nodes to the on-premise virtual_cloud.

Example: create_onprem_tag.yml

```

---
resources:
- data:
    name: onprem_cloud_tag
    uuid: 2eefeb06-2e7c-11e9-b210-d663bd873d93
    fq_name:
    - onprem_cloud_tag
    tag_type_name: label
    tag_value: onprem_cloud_provider
  kind: tag
  Operation: CREATE

```

3. Update the on-premise credential with the public cloud keypair reference.

- a. Use the UUID of the already created credential resource. List the credentials using the following requests to obtain the UUID.

```

contrailcli list contrail_control_node -d | grep -A 1 node_refs
contrailcli show node <uuidOfNodeRefsFromPreviousCommand> | grep -A 1 credential_refs

```

- b. In keypair_refs use the UUID of the keypair created as part of deploying public cloud in the previous topic “Deploy Public Cloud Infrastructure.”

Example: update_onprem_keypair.yml

```

---
resources:
- data:
    uuid: c8d9bf8e-2f4a-11e9-bfac-0050568a3bf0

```

```

    keypair_refs:
      - uuid: 4e77005b-b7ba-489b-9891-8472cee8ghts
    kind: credential
    operation: UPDATE

```

4. Create the on-premise `cloud_user` with the `credential_refs` pointing to the on-premise credential UUID, that was updated in Step "3" on page 158.

Example: `create_onprem_cloudduser.yml`

```

---
resources:
- data:
    uuid: 4e77005b-b7ba-489b-9891-aGFjawn9eadf
    name: onprem_cloud_user
    fq_name:
      - onprem_user
    credential_refs:
      - uuid: c8d9bf8e-2f4a-11e9-bfac-0050568a3bf0
    perms2:
      owner: admin
    kind: cloud_user
    operation: CREATE

```

5. Create the on-premise cloud objects.

Cloud object refers to `cloud_user` created in Step "4" on page 159 and the `virtual_cloud` reference tag created in Step "2" on page 158.

Example: `create_onprem_cloud_objects.yml`

```

---
resources:
- data:
    provisioning_state: CREATED
    uuid: dfb40e0d-c9f4-47cd-bd5c-MWVmZGQyOGZkNGZjCg
    name: onprem_cloud
    fq_name:
      - onprem_cloud
    perms2:

```

```

    owner: admin
  organization: juniper
  project: juniper-private
  cloud_user_refs:
    - uuid: 4e77005b-b7ba-489b-9891-aGFjawo9eadf
  kind: cloud
  operation: CREATE

```

```

- data:
  name: onprem_cloud_provider
  parent_type: cloud
  fq_name:
    - onprem_cloud
    - onprem_cloud_provider
  perms2:
    owner: admin
  type: private
  kind: cloud_provider
  operation: CREATE

```

```

- data:
  name: onprem_cloud_region
  parent_type: cloud-provider
  fq_name:
    - onprem_cloud
    - onprem_cloud_provider
    - onprem_cloud_region
  perms2:
    owner: admin
  kind: cloud_region
  operation: CREATE

```

```

- data:
  name: onprem_virtual_cloud
  parent_type: cloud-region
  fq_name:
    - onprem_cloud
    - onprem_cloud_provider
    - onprem_cloud_region
    - onprem_virtual_cloud
  perms2:
    owner: admin
  tag_refs:

```

```

- uuid: 2eefeb06-2e7c-11e9-b210-d663bd873d93
kind: virtual_cloud
operation: CREATE

- data:
  name: onprem_cloud_private_subnet
  uuid: 5ecfeb06-2e7c-11e9-b210-d663bd873d93
  parent_type: virtual-cloud
  fq_name:
  - onprem_cloud
  - onprem_cloud_provider
  - onprem_cloud_region
  - onprem_virtual_cloud
  - onprem_cloud_private_subnet
  perms2:
    owner: admin
  cidr_block: 192.168.1.0/24
kind: cloud_private_subnet
operation: CREATE

- data:
  name: onprem_cloud_private_subnet_gw
  uuid: 3defeb06-2e7c-11e9-b210-d663bd873d93
  parent_type: virtual-cloud
  fq_name:
  - onprem_cloud
  - onprem_cloud_provider
  - onprem_cloud_region
  - onprem_virtual_cloud
  - onprem_cloud_private_subnet_gw
  perms2:
    owner: admin
  cidr_block: 192.168.2.0/24
kind: cloud_private_subnet
operation: CREATE

```

6. Create the on-premise gateway node.

- tag_refs, credential_refs, and cloud_private_subnet_refs are the UUID of the respective resources created or updated in [Step "2" on page 158](#), [Step "3" on page 158](#), and [Step "5" on page 159](#) respectively.

- `cloud_private_subnet_refs` is specifically from the `cloud_private_subnet` created for the on-premise gateway.

Example: `create_onprem_mcgw_node.yml`

```
---
resources:
- data:
    uuid: 41f99f2d-a5a4-4e2c-b598-c173cf748953
    name: onprem_gateway
    type: private
    hostname: onprem_gateway
    ip_address: 10.87.74.132
    interface_name: eno1
    fq_name:
      - default-global-system-config
      - onpre_virtual_cloud
    parent_type: global-system-config
    tag_refs:
      - uuid: 2eefeb06-2e7c-11e9-b210-d663bd873d93
    credential_refs:
      - uuid: c8d9bf8e-2f4a-11e9-bfac-0050568a3bf0
    cloud_private_subnet_refs:
      - uuid: 3defeb06-2e7c-11e9-b210-d663bd873d93
    kind: node

# Create private interface for onprem gateway
- data:
    parent_type: node
    parent_uuid: 41f99f2d-a5a4-4e2c-b598-c173cf748953
    name: bond0
    ip_address: 192.168.2.1
    pxe_enabled: false
    kind: port
```

7. Create the on-premise `contrail_multicloud_gateway_node` role and update `parent_uuid` with `contrail_cluster` UUID.

Use the following request to get the contrail_cluster UUID:

```
contrailcli list contrail_cluster | grep uuid
```

Update node_refs UUID with the gateway node created earlier in this step.

Example: create_onprem_mcgw_node_role.yml

```
---
resources:
- data:
    name: onprem_contrail_multicloud_gw_node
    node_refs:
    - uuid: 41f99f2d-a5a4-4e2c-b598-c173cf748953
    protocols_mode:
    - ssl_client
    default_gateway: 192.168.2.254
    parent_type: contrail-cluster
    parent_uuid: a5063dde-2681-11e9-8021-0050568a3bf0
    kind: contrail_multicloud_gw_node
```

8. Update the on-premise compute and controller node.

Link the on-premise cluster nodes (compute/controller) to the virtual_cloud created for the on-premise cluster using tag. Use the UUID of the node object created using the UI as part of the Contrail cluster deployment.

Use the following request to get the node UUID:

```
contrailcli list contrail_control_node | grep uuid
contrailcli list contrail_vrouter_node | grep uuid
```

tag_refs, and cloud_private_subnet_refs are the UUID of the respective resources created or updated in [Step "2" on page 158](#) and [Step "5" on page 159](#).

Example: update_onprem_nodes.yml

```
---
```

```

resources:
#Link onprem cluster nodes to the virtaul_cloud created for onprem cluster
- data:
    uuid: c8d9d4ec-2f4a-11e9-bfac-0050568a3bf0
    cloud_private_subnet_refs:
    - uuid: 5ecfeb06-2e7c-11e9-b210-d663bd873d93
    tag_refs:
    - uuid: 2eefeb06-2e7c-11e9-b210-d663bd873d93
    kind: node
    operation: UPDATE

- data:
    uuid: c8d9c1b4-2f4a-11e9-bfac-0050568a3bf0
    cloud_private_subnet_refs:
    - uuid: 5ecfeb06-2e7c-11e9-b210-d663bd873d93
    tag_refs:
    - uuid: 2eefeb06-2e7c-11e9-b210-d663bd873d93
    kind: node
    operation: UPDATE

```

9. Update the on-premise cloud state with NOSTATE to trigger deployment of the on-premise cloud. Use the onprem_cloud objects UUID created in Step "5" on [page 159](#).

```

---
resources:
- data:
    provisioning_state: NOSTATE
    uuid: dfb40e0d-c9f4-47cd-bd5c-MWVmZGQyOGZkNGZjCg
    kind: cloud
    operation: UPDATE

```

Wait for the cloud deployment logs in **/var/log/contrail/cloud.log** to complete before proceeding to the next steps. When completed, the provisioning_state of the cloud resource changes from NOSTATE to either UPDATED or UPDATE_FAILED.

Extending On-Premise Contrail Cluster to Public Cloud

To extend the on-premise Contrail cluster to the public cloud:

1. Use the following request to get the cloud UUIDs.

```
contrailcli list cloud | grep uuid
```

2. Use the following request to get the UUID of the contrail_cluster.

```
contrailcli list contrail_cluster | grep uuid
```

3. Run the following request payload to extend the on-premise Contrail cluster to the public cloud.

Example: extend_onprem_to_coud.yml

```
---
resources:
- data:
    uuid: a5063dde-2681-11e9-8021-0050568a3bf0
    provisioning_state: NOSTATE
    provisioning_action: ADD_CLOUD
    cloud_refs:
    - uuid: dfb40e0d-c9f4-47cd-bd5c-MWVmZGQyOGZkNGZjCg
    - uuid: dfb40e0d-c9f4-47cd-bd5c-1efdd28fd4fc
    mc_gw_info:
      AS: 65000
      openvpn_port: 443
      vpn_lo_network: 100.65.0.0/16
      vpn_network: 100.64.0.0/16
      bfd_interval: 200ms
      bfd_multiplier: 5
      bfd_interval_multihop: 500ms
      bfd_multiplier_multihop: 5
    kind: contrail_cluster
    operation: UPDATE
```

With this request, you trigger the Contrail multicloud Ansible playbooks to start deploying Contrail roles on the public cloud, which includes the Contrail multicloud gateway role.

Wait for the cloud deployment logs in **/var/log/contrail/cloud.log** to complete before proceeding to the next steps. When completed, the `provisioning_state` of the cloud resource changes from `NOSTATE` to either `UPDATED` or `UPDATE_FAILED`.

RELATED DOCUMENTATION

[Deleting the Contrail MultiCloud Cluster](#) | 145

[Updating the Contrail Multicloud Cluster](#) | 142