Engineering
Simplicity

Contrail® Networking

Contrail Networking Service Provider Focused Features Guide

Published
2024-08-14

RELEASE
21.4

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

Contrail® Networking Contrail Networking Service Provider Focused Features Guide
21.4
Copyright © 2024 Juniper Networks, Inc. All rights reserved.

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

YEAR 2000 NOTICE

END USER LICENSE AGREEMENT

Table of Contents

About This Guide

Use this guide to understand the features that would be used by service providers. This guide also provides information about advanced service chain configuration in Contrail Networking.

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 for Container Networking Environments User Guide	Provides information about installing and using Contrail Networking in containerized environments using Kubernetes orchestration.
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 Monitoring and Troubleshooting Guide	Provides information about Contrail Insights and Contrail analytics.

RELATED DOCUMENTATION

README Access to Contrail Networking Registry 21xx

Contrail Networking Release Notes 21xx

Tungsten Fabric Architecture Guide

1

CHAPTER

Data Plane Optimization

Configuring the Data Plane Development Kit (DPDK) Integrated with Contrail vRouter

IN THIS SECTION

DPDK Support in Contrail

Contrail Networking supports the Data Plane Development Kit (DPDK). DPDK is an open source set of libraries and drivers for fast packet processing. DPDK enables fast packet processing by allowing network interface cards (NICs) to send direct memory access (DMA) packets directly into an application’s address space, allowing the application to poll for packets, and thereby avoiding the overhead of interrupts from the NIC.

Integrating with DPDK allows a Contrail vRouter to process more packets per second than is possible when running as a kernel module.

In Contrail Networking, before you use DPDK the DPDK configuration should be defined in `instances.yaml` for ansible based provision, or in `host.yaml` for helm-based provision. The AGENT_MODE configuration specifies whether the hypervisor is provisioned in the DPDK mode of operation.

When a Contrail compute node is provisioned with DPDK, the corresponding yaml file specifies the number of CPU cores to use for forwarding packets, the number of huge pages to allocate for DPDK, and the UIO driver to use for DPDK.

Preparing the Environment File for Provisioning a Cluster Node with DPDK

The environment file is used at provisioning to specify all of the options necessary for the installation of a Contrail cluster, including whether any node should be configured to use DPDK.

Each node to be configured with the DPDK vRouter must be listed in the provisioning file with a dictionary entry, along with the percentage of memory for DPDK huge pages and the CPUs to be used.

The following are descriptions of the required entries for the server configuration. :

- `HUGE_PAGES`—Specify the percentage of host memory to be reserved for the DPDK huge pages. The reserved memory will be used by the vRouter and the Quick Emulator (QEMU) for allocating memory resources for the virtual machines (VMs) spawned on that host.

  > **NOTE**: The percentage allocated to `HUGE_PAGES` should not be too high, because the host Linux kernel also requires memory.

  When using huge pages with DPDK, ensure that:

  - By default, 2MB huge pages are provisioned.
  - If 1GB huge pages are required for your node, set the 1GB huge page setting to your preference and disable 2MB hugepages. We strongly recommend setting the 2M huge page setting to 0 instead of deleting the variable to ensure the node does not attempt to use 2M hugepages.
  - Huge pages are set differently depending on environment, but you can accomplish setting the 2M huge pages to 0 in most environments by setting 2M huge pages to 0 in the ContrailDpdkParameters hierarchy within the contrail_services.yaml file.

- `CPU_CORE_MASK`—Specify a CPU affinity mask with which vRouter will run. vRouter will use only the CPUs specified for its threads of execution. These CPU cores will be constantly polling for packets, and they will be displayed as 100% busy in the output of “top”.

  The supported format is hexadecimal (for example, 0xf).

- `DPDK_UIO_DRIVER`—Specify the UIO driver to be used with DPDK.

  The supported values include:

  - `vfio-pci`—Specify that the vfio module in the Linux kernel should be used instead of uio. The vfio module protects memory access using the IOMMU when a SR-IOV virtual function is used as the physical interface of vrouter.
  - `uio_pci_generic`—Specify that the UIO driver built into the Linux kernel should be used. This option does not support the use of SR-IOV VFs. This is the default option if DPDK_UIO_DRIVER is not specified.
  - `mlnx` – For Mellanox ConnectX-4 and ConnectX-5 NICs.

NOTE: For RHEL and Intel x710 Fortville-based NIC, use `vfio-pci` instead of the default uio_pci_generic.

Use the standard Ansible or helm-based provision procedure. Upon completion, your cluster, with specified nodes using the DPDK vRouter implementation, is ready to use.

Sample configuration in instances.yml for ansible-based provision

```
Bms1:
    provider: bms
    ip: ip-address
    roles:
      vrouter:
                AGENT_MODE: dpdk
                CPU_CORE_MASK: “0xff”
                DPDK_UIO_DRIVER: uio_pci_generic
                HUGE_PAGES: 32000
```

Sample configuration in host.yml for helm-based provision

```
...
AGENT_MODE: dpdk
CPU_CORE_MASK: “0xff”
DPDK_UIO_DRIVER: uio_pci_generic
HUGE_PAGES: 32000
```

Creating a Flavor and Understanding Huge Pages for DPDK

To launch a VM in a DPDK-enabled vRouter hypervisor, the flavor for the VM should be set to use huge pages. The use of huge pages is a requirement for using a DPDK vRouter.

Use the following command to add the flavor, where `m1.large` is the name of the flavor. When a VM is created using this flavor, OpenStack ensures that the VM will only be spawned on a compute node that has huge pages enabled.

```
# openstack flavor set m1.large --property hw:mem_page_size=large
```

Huge pages are enabled for compute nodes where vRouter is provisioned with DPDK.

If a VM is spawned with a flavor that does not have huge pages enabled, the VM should not be created on a compute node on which vRouter is provisioned with DPDK.

You can use OpenStack availability zones or host aggregates to exclude the hosts where vRouter is provisioned with DPDK.

Note the following when using huge pages with DPDK:

- The size of the flow table on your node should dictate the size of your huge page configuration. Use 1GB huge pages in environments with more flow table entries.
- By default, 2MB huge pages are provisioned.
- If 1GB huge pages are required for your node, set the 1GB huge page setting to your preference and disable 2M hugepages. We strongly recommend setting the 2M huge page setting to 0 instead of deleting the variable to ensure the environment does not attempt to use 2M hugepages.

  Huge pages are set differently depending on environment, but you can accomplish setting the 2M huge pages to 0 in most environments by setting 2M huge pages to 0 in the ContrailDpdkParameters hierarchy within the contrail_services.yaml file.

Configuring and Verifying MTU for DPDK vRouter

This section describes how you configure the maximum transmission unit (MTU) for DPDK vRouter. To set MTU, you need to specify the desired value for `mtu` in the **contrail_vrouter_dpdk_bond.yaml** file.

```
network_config:
  -
    type: contrail_vrouter_dpdk
    name: vhost0
    members:
      -
        type: interface
        name: em3
      -
        type: interface
        name: em1
    mtu: 9100
    bond_mode: 2
    bond_policy: 802.3ad
```

You can verify the configured value from hypervisor by running the following command:

```
$ ip link list vhost0
39: vhost0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9100 qdisc pfifo_fast state UNKNOWN mode
DEFAULT group default qlen 1000
    link/ether 98:03:9b:a7:3b:a0 brd ff:ff:ff:ff:ff:ff
```

You can use the `vif -g` or `vif --get` command to view the status of the bond interfaces in a DPDK vRouter.

For example,

```
# vif --get 0
Vrouter Interface Table

[...]
vif0/0      PCI: 0000:00:00.0 (Speed 20000, Duplex 1) NH: 4
            Type:Physical HWaddr:00:1b:21:bb:f9:48 IPaddr:0.0.0.0
            Vrf:0 Mcast Vrf:65535 Flags:TcL3L2VpEr QOS:-1 Ref:26
            RX device packets:668852  bytes:110173140 errors:0
            RX port   packets:207344 errors:0
            RX queue errors to lcore 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
            Fabric Interface: eth_bond_bond0  Status: UP  Driver: net_bonding
            Slave Interface(o): 0000:02:00.0  Status: UP  Driver: net_ixgbe
            Slave Interface(1): 0000:02:00.1  Status: UP  Driver: net_ixgbe
            Vlan Id: 101  VLAN fwd Interface: bond
            RX packets:207344  bytes:45239337 errors:0
            TX packets:326159  bytes:237905360 errors:4
            Drops:0
            TX port   packets:326145 errors:10
            TX device packets:915402  bytes:511551768 errors:0
```

RELATED DOCUMENTATION

Configuring Single Root I/O Virtualization (SR-IOV)

Monitoring Bond Interfaces in DPDK Enabled Devices

vRouter Command Line Utilities

http://www.dpdk.org

Apply non 1500 byte default MTU to VNs

Contrail Networking Release 21.4.L3 supports and implements MTU property within tenant and virtual network resources based on the priority from highest to lowest as follows:

1. Explicitly defined virtual network MTU, if provided
2. Per-tenant defined default MTU, if virtual network MTU is not defined explicitly
3. Default 1500 bytes, if no MTU is set per-tenant

The per-tenant MTU value can be configured and updated without impacting Contrail's current default behaviour. The default MTU per-tenant is 'None' and a default 1500 bytes MTU is assigned to any new virtual network for which no new explicit MTU is defined.

If a virtual network is created with zero MTU, then the MTU specified at the time of creating virtual network is is assigned. In case, MTU is not specified then its default value is applied.

NOTE: Only Contrail Command user interface supports this feature.

Note: If a VN is created with MTU as 0, project MTU will be assigned to VN if specified else default MTU will be assigned

Configuring Single Root I/O Virtualization (SR-IOV)

IN THIS SECTION

Overview: Configuring SR-IOV

Contrail Networking supports single root I/O virtualization (SR-IOV) on Ubuntu systems and on Red Hat Enterprise Linux (RHEL) operating systems as well.

SR-IOV is an interface extension of the PCI Express (PCIe) specification. SR-IOV allows a device, such as a network adapter to have separate access to its resources among various hardware functions.

As an example, the Data Plane Development Kit (DPDK) library has drivers that run in user space for several network interface cards (NICs). However, if the application runs inside a virtual machine (VM), it does not see the physical NIC unless SR-IOV is enabled on the NIC.

This topic shows how to configure SR-IOV with your Contrail Networking system.

Enabling ASPM in BIOS

To use SR-IOV, it must have Active State Power Management (ASPM) enabled for PCI Express (PCIe) devices. Enable ASPM in the system BIOS.

NOTE: The BIOS of your system might need to be upgraded to a version that can enable ASPM.

Configuring SR-IOV Using the Ansible Deployer

You must perform the following tasks to enable SR-IOV on a system.

1. Enable the Intel Input/Ouput Memory Management Unit (IOMMU) on Linux.
2. Enable the required number of Virtual Functions (VFs) on the selected NIC.
3. Configure the names of the physical networks whose VMs can interface with the VFs.
4. Reboot Nova compute.

   ```
   service nova-compute restart
   ```

5. Configure a Nova Scheduler filter based on the new PCI configuration, as in the following example:

```
/etc/nova/nova.conf
     [default]
     scheduler_default_filters = PciPassthroughFilter
     scheduler_available_filters = nova.scheduler.filters.all_filters
     scheduler_available_filters =
nova.scheduler.filters.pci_passthrough_filter.PciPassthroughFilter
```

6. Restart Nova Scheduler.

```
service nova-scheduler restart
```

The above tasks are handled by the Ansible Deployer playbook. The cluster members and its configuration parameters are specified in the **instances.yaml** file located in the config directory within the ansible-deployer repository.

The compute instances that are going to be in SR-IOV mode should have an SR-IOV configuration. The **instance.yaml** snippet below shows a sample instance definition.

```
instances:
  bms1:
    provider: bms
    ip: ip-address
    roles:
      openstack:
  bms2:
    provider: bms
    ip:ip-address
    roles:
      config_database:
      config:
      control:
      analytics_database:
      analytics:
      webui:
  bms3:
    provider: bms
    ip: ip-address
    roles:
```

```
    openstack_compute:
    vrouter:
      SRIOV: true
      SRIOV_VF: 3
      SRIOV_PHYSICAL_INTERFACE: eno1
      SRIOV_PHYS_NET:  physnet1
```

Configuring SR-IOV Using Helm

You must perform the following tasks to enable SR-IOV on a system.

1. Enable the Intel Input/Ouput Memory Management Unit (IOMMU) on Linux.
2. Enable the required number of Virtual Functions (VFs) on the selected NIC.
3. Configure the names of the physical networks whose VMs can interface with the VFs.
4. Reboot Nova compute.

   ```
   service nova-compute restart
   ```

5. Configure a Nova Scheduler filter based on the new PCI configuration, as in the following example:

   ```
   /etc/nova/nova.conf
         [default]
         scheduler_default_filters = PciPassthroughFilter
         scheduler_available_filters = nova.scheduler.filters.all_filters
         scheduler_available_filters =
         nova.scheduler.filters.pci_passthrough_filter.PciPassthroughFilter
   ```

6. Restart Nova Scheduler.

   ```
   service nova-scheduler restart
   ```

The above tasks are handled by the Helm charts. The cluster members and its configuration parameters are specified in the **multinode-inventory** file located in the config directory within the openstack-helm-infra repository.

For Helm, the configuration and SR-IOV environment-specific parameters must be updated in three different places:

- The compute instance must be set as contrail-vrouter-sriov.

  For example, the following is a snippet from the **tools/gate/devel/multinode-inventory.yaml** file in the openstack-helm-infra repository.

```
all:
  children:
    primary:
      hosts:
        node1:
          ansible_port: 22
          ansible_host: host-ip-address
          ansible_user: ubuntu
          ansible_ssh_private_key_file: /home/ubuntu/.ssh/insecure.pem
          ansible_ssh_extra_args: -o StrictHostKeyChecking=no
  nodes:
    children:
      openstack-compute:
        children:
           contrail-vrouter-sriov: #compute instance set to contrail-vrouter-sriov
              hosts:
                node7:
                ansible_port: 22
                 ansible_host: host-ip-address
                 ansible_user: ubuntu
                 ansible_ssh_private_key_file: /home/ubuntu/.ssh/insecure.pem
                 ansible_ssh_extra_args: -o StrictHostKeyChecking=no
```

- Contrail-vrouter-sriov must be labeled appropriately.

  For example, the following is a snippet from the **tools/gate/devel/multinode-vars.yaml** in the openstack-helm-infra repository.

```
nodes:
  labels:
    primary:
    - name: openstack-helm-node-class
      value: primary
```

```
              all:
           - name: openstack-helm-node-class
               value: general
           contrail-controller:
           - name: opencontrail.org/controller
                 value: enabled
           openstack-compute:
           - name: openstack-compute-node
                 value: enabled
              contrail-vrouter-dpdk:
              - name: opencontrail.org/vrouter-dpdk
                    value: enabled
               contrail-vrouter-sriov: # label as contrail-vrouter-sriov
               - name: vrouter-sriov
                   value: enabled
```

- SR-IOV config parameters must be updated in the **contrail-vrouter/values.yaml** file.

  For example, the following is a snippet from the **contrail-vrouter/values.yaml** file in the contrail-helm-deployer repository.

```
contrail_env_vrouter_kernel:
  AGENT_MODE: kernel

contrail_env_vrouter_sriov:
  SRIOV: true
  per_compute_info:
    node_name: k8snode1
    SRIOV_VF:  10
    SRIOV_PHYSICAL_INTERFACE: enp129s0f1
    SRIOV_PHYS_NET:  physnet1
```

Launching SR-IOV Virtual Machines

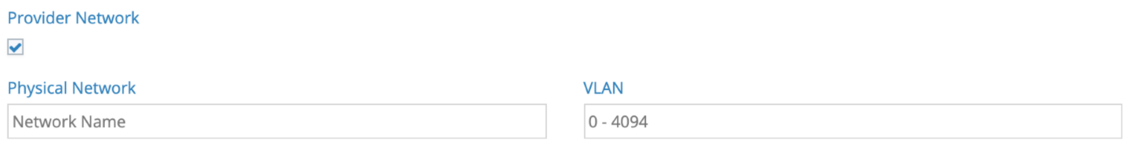
After ensuring that SR-IOV features are enabled on your system, use one of the following procedures to create a virtual network from which to launch an SR-IOV VM, either by using the Contrail Web UI or the CLI. Both methods are included.

Using the Contrail Web UI to Enable and Launch an SR-IOV Virtual Machine

To use the Contrail Web UI to enable and launch an SR-IOV VM:

1. At **Configure > Networking > Networks**, create a virtual network with SR-IOV enabled. Ensure the virtual network is created with a subnet attached. In the Advanced section, select the **Provider Network** check box, and specify the physical network already enabled for SR-IOV (in `testbed.py` or `nova.conf`) and its VLAN ID. See Figure 1 on page 13.

   Figure 1: Edit Network

   Provider Network

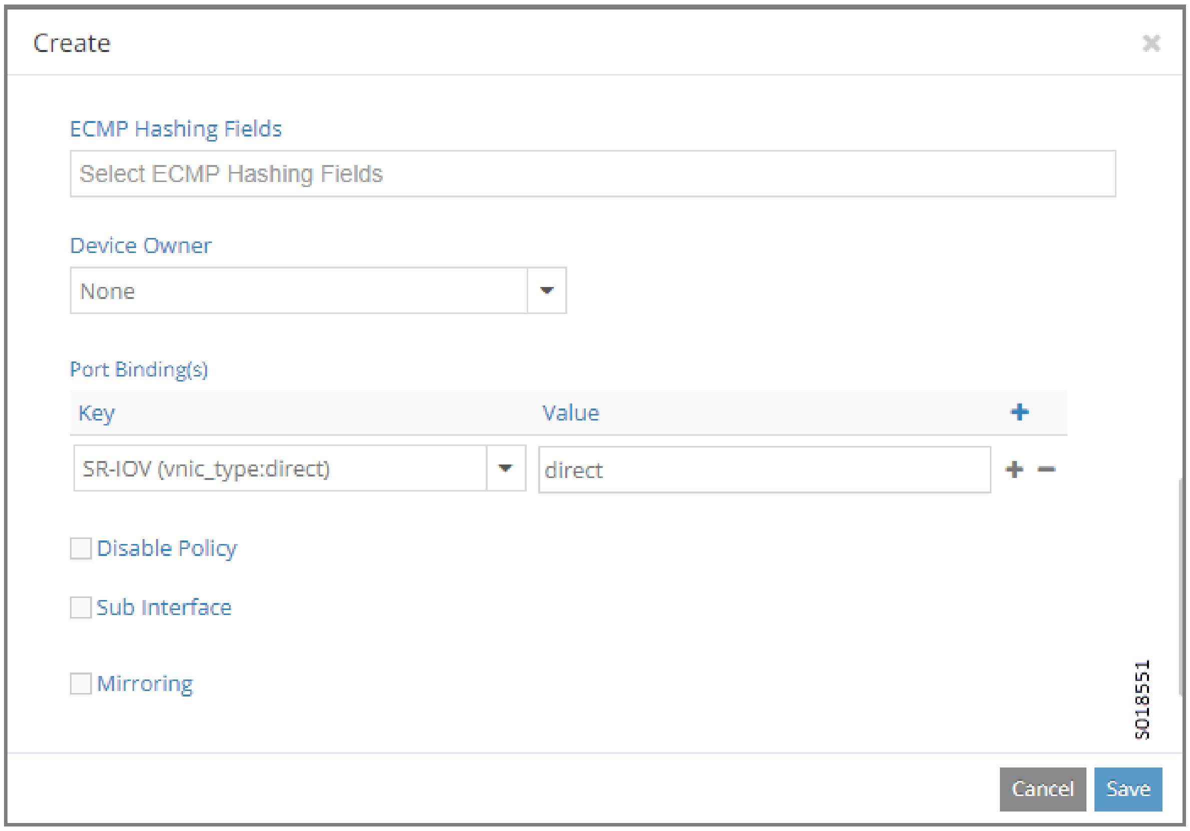
   ✓

   Physical Network: Network Name

   VLAN: 0 - 4094

2. On the virtual network, create a Neutron port (**Configure > Networking > Ports**), and in the **Port Binding** section, define a **Key** value of SR-IOV and a **Value** of direct. See Figure 2 on page 14.

Figure 2: Create Port


Create

ECMP Hashing Fields: Select ECMP Hashing Fields

Device Owner: None

Port Binding(s)

Key	Value
SR-IOV (vnic_type:direct)	direct

☐ Disable Policy

☐ Sub Interface

☐ Mirroring

3. Using the UUID of the Neutron port you created, use the `nova boot` command to launch the VM from that port.

```
nova boot --flavor m1.large --image <image name> --nic port-id=<uuid of above port> <vm name>
```

Using the CLI to Enable and Launch SR-IOV Virtual Machines

To use CLI to enable and launch an SR-IOV VM:

1. Create a virtual network with SR-IOV enabled. Specify the physical network already enabled for SR-IOV (in `testbed.py` or `nova.conf`) and its VLAN ID.

   The following example creates `vn1` with a VLAN ID of 100 and is part of `physnet1`:

   ```
   neutron net-create --provider:physical_network=physnet1 --provider:segmentation_id=100 vn1
   ```

2. Create a subnet in vn1.

   ```
   neutron subnet-create vn1 a.b.c.0/24
   ```

3. On the virtual network, create a Neutron port on the subnet, with a binding type of direct.

```
neutron port-create --fixed-ip subnet_id=<subnet uuid>,ip_address=<IP address from above subnet> --name <name
of port> <vn uuid> --binding:vnic_type direct
```

4. Using the UUID of the Neutron port created, use the `nova boot` command to launch the VM from that port.

   ```
   nova boot --flavor m1.large --image <image name> --nic port-id=<uuid of above port> <vm name>
   ```

5. Log in to the VM and verify that the Ethernet controller is VF by using the `lspci` command to list the PCI buses.

   The VF that gets configured with the VLAN can be observed using the `ip link` command.

RELATED DOCUMENTATION

Configuring the Data Plane Development Kit (DPDK) Integrated with Contrail vRouter | 2

Optimizing DPDK vRouter Performance Through Full CPU Partitioning and Isolation

Contrail Networking Release 2003 supports full CPU partitioning. CPU isolation is an RHEL method to partition and isolate the CPU cores on a compute node from the symmetric multiprocessing (SMP) balancing and scheduler algorithms. The full CPU isolation feature optimizes the performance of DPDK vRouter when deployed with the DPDK settings recommended for RHOSP.

CPU isolation helps isolate forwarding cores, VNF cores, and service cores so that VNF threads and service threads do not send processing requests to forwarding cores. By applying CPU isolation, you can allocate dedicated forwarding cores to the DPDK VM and ensure that other processes do not send processing requests to the cores allocated to DPDK vRouter, which in turn improves the performance of vRouter to a large extent.

For CPU isolation and partitioning, RedHat recommends two methods. The first method is by using a utility called `tuned`, which partitions the CPU to virtual network functions (VNFs) and isolates these cores from the host OS. The `tuned` method removes isolated CPUs from the common CPU list that is used to process all tasks and perform CPU isolation after the system boot, by using the `systemd` process.

The second is `isolcpus`, a kernel parameter that keeps CPUs away from the Linux scheduler. Similar to `tuned`, the `isolcpus` method also removes isolated CPUs from the common CPU list that is used to process all tasks, and performs CPU isolation at system startup. To enable `isolcpus`, you need to modify the

GRUB configuration in **/etc/default/grub** so that a new set of isolated CPU is considered. The node needs to be restarted for the changes to take effect.

To enable CPU isolation using `tuned`, configure the `ContrailDpdkParameters` in **/tripleo-heat-templates/environments/contrail/contrail-services.yaml** for RHOSP and `SERVICE_CORE_MASK` and `DPDK_CTRL_THREAD_MASK` parameters in **/vrouter/agent-dpdk/entrypoint.sh** file for Contrail Ansible Deployer.

In contrail-services.yaml

```
# Tuned-d profile configuration
#   TunedProfileName -  Name of tuned profile
#   IsolCpusList     -  Logical CPUs list to be isolated from the host process (applied via cpu-
partitioning tuned).
#                       It is mandatory to provide isolated cpus for tuned to achive optimal
performance.
#                       Example: "3-8,12-15,18"
# These paramters are to be set per a role, e.g.:
#  ComputeParameters:
#    TunedProfileName: "cpu-partitioning"
#    IsolCpusList: "3-8,12-15,20"
#  ContrailDpdkParameters:
#    TunedProfileName: "cpu-partitioning"
#    IsolCpusList: "3-20"
#  ContrailSriovParameters:
#    TunedProfileName: "cpu-partitioning"
#    IsolCpusList: "3-20"
```

In entrypoint.sh

```
# Cpu coremask for DPDK
# - forwarding threads pinning
#CPU_CORE_MASK='0x01'
# - service threads pinning
#SERVICE_CORE_MASK=''
# - dpdk ctrl threads pinning
#DPDK_CTRL_THREAD_MASK=''
```

To configure `isolcpus`, modify the following parameters in GRUB:

```
ContrailDpdkParameters:
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt intel_iommu=on
```

```
isolcpus=3-20"
```

The `isolcpu` tuning needs to be done for VNFs (VM) as well. This is to ensure that the VM can protect and isolate the Poll Mode Driver (PMD) thread cores from CPU usage by other processes. On Centos and RHEL, CPU tuning is done by using the utilities `isolcpu` and `tuned`.

Contrail DPDK vRouter Support for Intel DDP Technology in Fortville NICs

In Contrail Networking Release 2011, the Contrail DPDK enabled vRouter uses the Intel dynamic device personalization (DDP) technology. The Intel DDP technology provides a programmable pipeline, which enables you to meet specific use cases as per your requirement. Similarly in Contrail Networking, the Intel DDP technology enables you to forward packets with MPLSoGRE encapsulation. The Intel DDP technology is supported only by Intel Ethernet 700 Series (Fortville Series) NICs.

In previous releases, Intel Ethernet 500 Series (Niantic Series) NICs are unable to perform load-balancing for MPLSoGRE packets as they lack in port information. Due to this, all incoming packets between a pair of compute nodes start lining up in the same hardware receiving queue (Rx) of the NIC. The vRouter performs software load-balancing by distributing the packets to other CPU cores, which processes the packets in the Rx queue. This reduces the capacity of the vRouter and affects its performance.

In release 2011, Contrail Networking uses the Intel DDP technology, which enables the Fortville NICs to perform load-balancing for the MPLSoGRE packets. The Intel DDP technology allows dynamic re-configuration of the packet processing pipeline in the NIC at runtime, without rebooting the server. Contrail Networking is configured with an MPLSoGRE profile to process the incoming packets with MPLSoGRE encapsulation. The MPLSoGRE profile enables the NIC to distribute the packets evenly across different hardware Rx queues and enables the CPU cores to perform proportionally. This increases the performance of the vRouter.

Starting from release 2011, you can use the following commands to enable, add, delete, or view the list of DDP profiles loaded in a DPDK enabled vRouter:

- The Intel DDP profile is not enabled by default in DPDK enabled vRouter. You can pass `--ddp` as a command line argument when the system comes up to enable DDP in DPDK enabled vRouters for Fortville NICs.

- Alternatively, during runtime you can execute the `dpdkconf -ddp add` command present in the contrail-tools container to enable DDP in vRouter for Fortville NICs.

```
(contrail-tools)[root@cs-scale-02 /]$ dpdkconf --ddp add
Programming DDP image mplsogreudp - success
```

- Use the `dpdkconf --ddp delete` command on the CLI to remove a DDP profile already loaded in the vRouter for Fortville NICs.

```
(contrail-tools)[root@cs-scale-02 /]$ dpdkconf --ddp delete
vr_dpdk_ddp_del: Removed DDP image mplsogreudp - success
```

- Use the `dpdkinfo --ddp list` command on the CLI to display the list of DDP profiles loaded in the vRouter for Fortville NICs.

```
(contrail-tools)[root@cs-scale-02 /]$ dpdkinfo --ddp list
Profile count is: 1

Profile 0:
Track id:     0x8000000c
Version:      1.0.0.0
Profile name: L2/L3 over MPLSoGRE/MPLSoUDP
(contrail-tools)[root@cs-scale-02 /]$
```

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
2011	In Contrail Networking Release 2011, the Contrail DPDK enabled vRouter uses the Intel dynamic device personalization (DDP) technology. The Intel DDP technology provides a programmable pipeline, which enables you to meet specific use cases as per your requirement

RELATED DOCUMENTATION

vRouter Command Line Utilities

Contrail vRouter MAC Address - IP Address Learning and Bidirectional Forwarding and Detection Health Checking for Pods on Virtual Machines

In Contrail Networking Release 2011, the Contrail vRouter agent dynamically learns the MAC address-IP address binding of a pod deployed on a virtual machine (VM). This enables the vRouter agent to perform an efficient pod to pod communication in Contrail Networking.

In previous releases, the MAC address - IP address of a pod is assigned by OpenStack. Contrail Networking is unable to perform pod to pod communication as it does not have the reachability information of the pods hosted by the VMs.

Starting in Contrail Networking Release 21.3.1, a vRouter can learn multiple MAC-IP address bindings for a single MAC address when **Dynamic Address Learning** is enabled in a virtual network. In previous Contrail Networking releases, a vRouter could only learn a single MAC-IP address binding per MAC address.

The Contrail vRouter automatically learns multiple MAC-IP address bindings when a single MAC address is bound to multiple IP addresses and **Dynamic Address Learning** is enabled; no additional user configuration to support learning of the MAC-IP address bindings is needed or possible. Cloud-networking environments using Openstack orchestration and Contrail Networking can seamlessly support the learning of IPVLAN MAC address-IP address bindings over VM interfaces as a result of the vRouter's ability to learn multiple MAC-IP address bindings to a single MAC address.

Starting in Contrail Networking Release 21.4.L4, the Contrail vRouter supports Dynamic Address Learning of MAC-IP address bindings for IPv6 address family as well.

In the Contrail Command user interface (UI), the **Dynamic Address Learning** checkbox must be enabled while creating a virtual network. This enables the vRouter agent to learn the MAC address-IP address of the pods connected to the virtual network.

In release 2011, Contrail Networking also supports Bidirectional Forwarding and Detection (BFD) based health check to verify the liveliness of a pod. In the Contrail Command user interface (UI), you must create a BFD health check service, where the **Health Check Type** is assigned as **VN IP List**. The BFD session is enabled for a list of target IP addresses.

In release 2011, Contrail Networking supports IPv4 target IP addresses and starting from Contrail Networking Release 21.4.L4, Contrail supports IPv6 target IP addresses as well. The vRouter agent learns these IP addresses through the MAC address - IP address learning feature. The BFD health check session is initiated, when the vRouter agent learns the target IP address assigned to the BFD health check service. The BFD health check monitors the target list of health check for newly learnt IP addresses. If the BFD session is detected as **DOWN**, the vRouter agent deletes the routes generated for the MAC address - IP address of a pod learned by the vRouter.

The vRouter agent also sends Address Resolution Protocol (ARP)/Neighbor Discovery Protocol (NDP) packets in regular intervals to the newly learnt IP addresses. The vRouter agent performs this action to check a pod's liveliness. If a pod responds to the ARP/NDP request sent by the vRouter, the pod is considered as **UP**. If the pod does not respond to the ARP/NDP packets, the pod is considered as **DOWN**. If vRouter identifies a pod as **DOWN**, it deletes the routes generated for the respective MAC address-IP address of the pod.

You must perform the following steps to enable the vRouter to dynamically learn the MAC address - IP address of a pod:

1. Navigate to **Overlay** > **Virtual Networks** page. Click **Create** to create a new virtual network.

   Alternatively, you can also edit the properties of an existing virtual network. To edit an existing virtual network, select a virtual network from the displayed list and click the **Edit (pencil)** icon.
2. Follow the steps given *Create Virtual Network* to create a virtual network.
3. In the **Create Virtual Network** page, select **Dynamic Address Learning** to enable vRouter to learn the MAC address - IP address of pods dynamically.
4. Click **Create** to create a VN where the vRouter can learn the MAC address - IP address of the pods connected to the VN.

   The **Virtual Networks** page is displayed listing the newly created virtual network.

You must perform the following steps to enable BFD based healthcheck for the pods deployed on a VM:

1. Navigate to **Services** > **Health Check**.
2. Click **Create** to create a new BFD based health check service.

   NOTE: If you want to edit the properties of an existing health check service, select a health check service from the displayed list and click the **Edit (pencil)** icon.
3. Enter values in the **Create Health Check Service** page according to the guidelines provided in Table 2 on page 21.
4. Click **Create**.

   The **Health Check** tab is displayed listing the newly created health check service.
5. Associate the newly created health check service to a virtual network.

Table 2: Create Health Check Fields

Field	Description
Name	Enter a name for the health check service you are creating.
Health Check Type	Select **VN IP List** to run health check on the IP addresses of the virtual networks.
Protocol	Protocol is set to **BFD** by default when **VN IP List** is selected as the**Health Check Type**. BFD health check enables you to verify pod liveliness.
Add all option	Select this to run BFD health check for all IP addresses learned by the vRouter Agent from learning the MAC address - IP address of a pod.
Target IP List	Select IP addresses from list to run BFD health check on the selected IP addresses.
Desired Min Tx Interval (milli secs)	Enter the desired minimum transmission (Tx) interval before transmitting BFD packets.
Required Min Rx Interval (milli secs)	Enter the minimum interval between successive BFD packets that is supported by the system.
Multiplier	Enter the number of BFD packets that must be missed successively from the remote end to declare the BFD session as **DOWN**.

RELATED DOCUMENTATION

Service Instance Health Checks | **84**

2

CHAPTER

Advanced Network Topologies

Configuring Virtual Networks for Hub-and-Spoke Topology

IN THIS SECTION

Contrail Networking supports hub-and-spoke topology, which can be used to ensure that virtual machines (VMs) don’t communicate with each other directly; their communication is only allowed indirectly by means of a designated hub virtual network.

Route Targets for Virtual Networks in Hub-and-Spoke Topology

Hub-and-spoke topology can be used to ensure that virtual machines (VMs) don’t communicate with each other directly; their communication is only allowed indirectly by means of a designated hub virtual network (VN). The VMs are configured in spoke VNs.

This is useful for enabling VMs in a spoke VN to communicate by means of a policy or firewall, where the firewall exists in a hub site.

hub-and-spoke topology is implemented using two route targets (`hub-rt` and `spoke-rt`), as follows:

- Hub route target (`hub-rt`):
  - The hub VN *exports* all routes tagged with `hub-rt`.
  - The spoke VN *imports* routes tagged with `hub-rt`, ensuring that the spoke VN has only routes exported by the hub VN.
  - To attract spoke traffic, the hub VN readvertises the spoke routes or advertises the default route.
- Spoke route target (`spoke-rt`):
  - All spoke VNs export routes with route target `spoke-rt`.

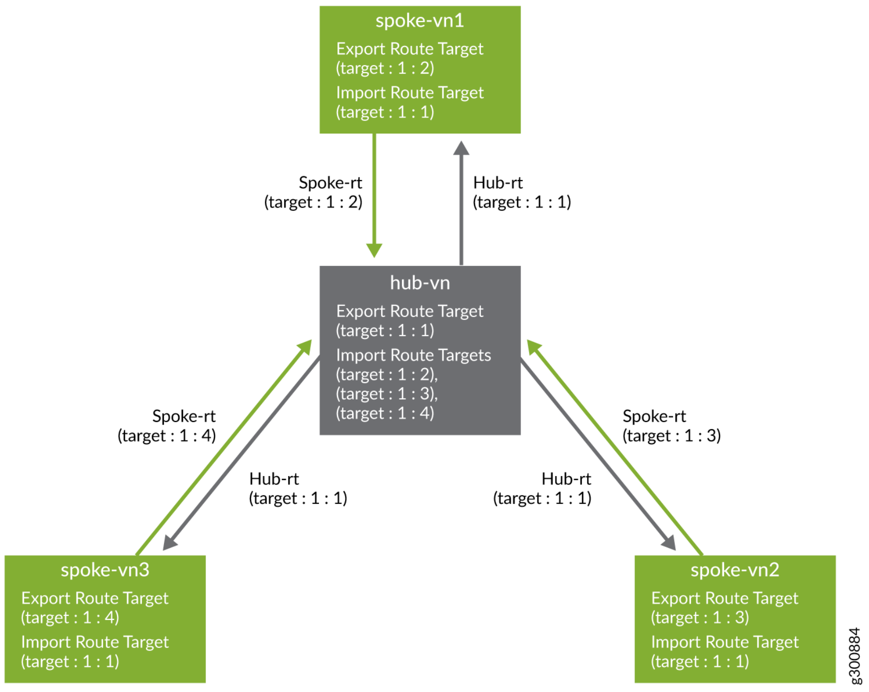
- The hub VN imports all spoke routes, ensuring that hub VN has all spoke routes.

NOTE: The hub VN or VRF can reside in an external gateway, such as an MX Series router, while the spoke VN resides in the Contrail controller.

Example: Hub-and-Spoke Topology

In the example shown in Figure 3 on page 24, the `hub-vn` is configured as a hub virtual network, and the three `spoke-vn`s are configured as spoke virtual networks. The hub and spokes each use a unique export route target. The `hub-vn` exports its `hub-rt (target:1:1)` routes to the spokes, and each `spoke-vn` imports them. Each `spoke-vn` exports its `spoke-rt (target:1:2, target:1:3, target:1:4)` routes to the hub, and the `hub-vn` imports them.

Figure 3: Hub-and-Spoke Topology



Troubleshooting Hub-and-Spoke Topology

The following examples provide methods to help you troubleshoot hub-and-spoke configurations.

Example: Validating the Configuration on the Virtual Network

The following example uses the api-server HTTP get request to validate the configuration on the virtual network.

Hub VN configuration:

```
curl -u admin:<password> http://<host ip>/virtual-network/<hub-vn-uuid>| python -m json.tool
```

```
{
    "virtual-network": {
        "display_name": "hub-vn",
        "fq_name": [
            "default-domain",
            "admin",
            "hub-vn"
        ],
        "export_route_target_list": {
            "route_target": [
                "target:1:2"
            ]
        },
        "import_route_target_list": {
            "route_target": [
                "target:1:1"
            ]
        },
    }
}
```

Spoke VN configuration:

```
curl -u admin:<password> http://<host ip>:8095/virtual-network/<spoke-vn-uuid> | python -m json.tool
```

```
{
{
    "virtual-network": {
        "display_name": "spoke-vn1",
        "fq_name": [
            "default-domain",
            "admin",
            "spoke-vn1"
        ],
        "export_route_target_list": {
            "route_target": [
                "target:1:1"
            ]
        },
        "import_route_target_list": {
            "route_target": [
                "target:1:2"
            ]
        },
    }
}
```

Example: Validate the Configuration on the Routing Instance

The following example uses `api-server HTTP get` request to validate the configuration on the routing instance.

Spoke VRF configuration (with a system-created VRF by schema transformer):

```
user@node:/opt/contrail/utils# curl -u admin:<password> http://<host ip>:8095/routing-instance/<spoke-vrf-uuid>|
python -m json.tool
```

```
{
    "routing-instance": {
        "display_name": "spoke-vn1",
        "fq_name": [
            "default-domain",
            "admin",
            "spoke-vn1",
            "spoke-vn1"
        ],
        "route_target_refs": [
            {
                "attr": {
                    "import_export": "export"
                },
                "href": "http://<host ip>:8095/route-target/446a3bbe-f263-4b58-
a537-8333878dd7c3",
                "to": [
                    "target:1:1"
                ],
                "uuid": "446a3bbe-f263-4b58-a537-8333878dd7c3"
            },
            {
                "attr": {
                    "import_export": null
                },
                "href": "http://<host ip>:8095/route-target/7668088d-
e403-414f-8f5d-649ed80e0689",
                "to": [
                    "target:64512:8000012"
                ],
                "uuid": "7668088d-e403-414f-8f5d-649ed80e0689"
            },
            {
                "attr": {
                    "import_export": "import"
                },
                "href": "http://<host ip>:8095/route-target/8f216064-8488-4486-8fce-
b4afb87266bb",
                "to": [
                    "target:1:2"
                ],
                "uuid": "8f216064-8488-4486-8fce-b4afb87266bb"
            }
        ],
        "routing_instance_is_default": true,
    }
}
```

Hub VRF configuration:

```
curl -u admin:<password> http://<host ip>:8095/routing-instance/<hub-vrf-uuid> | python -m json.tool
```

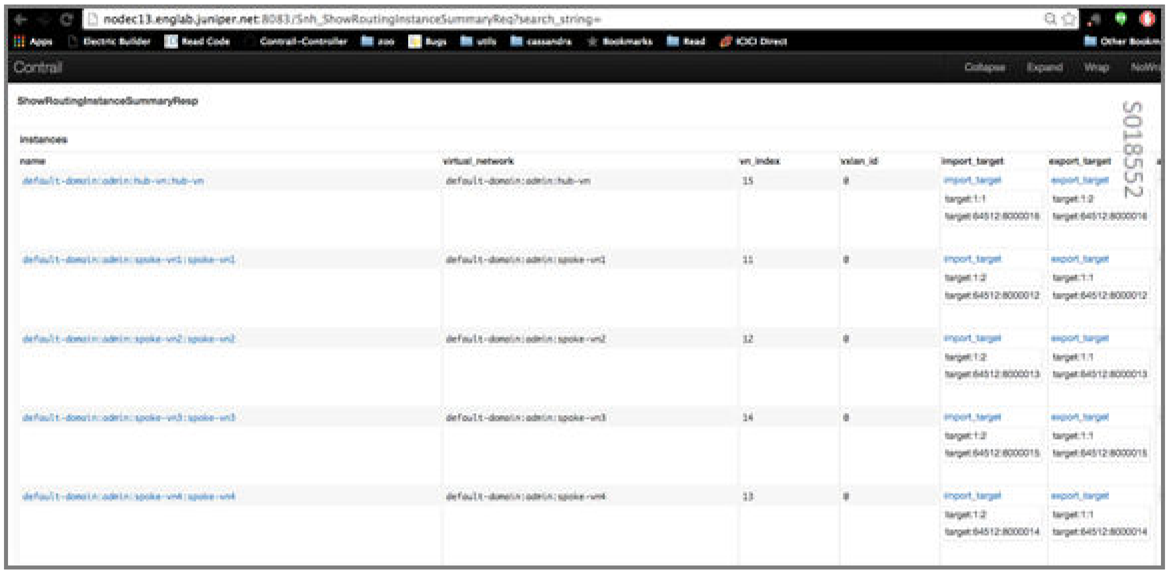
```
{
    "routing-instance": {
        "display_name": "hub-vn",
        "fq_name": [
            "default-domain",
            "admin",
            "hub-vn",
            "hub-vn"
        ],
        "route_target_refs": [
            {
                "attr": {
                    "import_export": "import"
                },
                "href": "http://<host ip>:8095/route-target/446a3bbe-f263-4b58-
a537-8333878dd7c3",
                "to": [
                    "target:1:1"
                ],
                "uuid": "446a3bbe-f263-4b58-a537-8333878dd7c3"
            },
            {
                "attr": {
                    "import_export": "export"
                },
                "href": "http://<host ip>:8095/route-target/8f216064-8488-4486-8fce-
b4afb87266bb",
                "to": [
                    "target:1:2"
                ],
                "uuid": "8f216064-8488-4486-8fce-b4afb87266bb"
            },
            {
                "attr": {
                    "import_export": null
                },
                "href": "http://<host ip>:8095/route-target/a85fec19-eed2-430c-
af23-9919aca1dd12",
                "to": [
                    "target:64512:8000016"
                ],
                "uuid": "a85fec19-eed2-430c-af23-9919aca1dd12"
            }
        ],
        "routing_instance_is_default": true,
    }
```

```
}
```

Example: Using Contrail Control Introspect

Figure 4 on page 29 shows the import and export targets for `hub-vn` and `spoke-vns`, by invoking `contrail-control-introspect`.

Figure 4: Contrail Introspect



ShowRoutingInstanceSummaryResp

instances

name	virtual_network	vn_index	vxlan_id	import_target	export_target
default-domain:admin:hub-vn:hub-vn	default-domain:admin:hub-vn	15	0	import_target target:1:1 target:64512:8000016	export_target target:1:2 target:64512:8000016
default-domain:admin:spoke-vn1:spoke-vn1	default-domain:admin:spoke-vn1	11	0	import_target target:1:2 target:64512:8000012	export_target target:1:1 target:64512:8000012
default-domain:admin:spoke-vn2:spoke-vn2	default-domain:admin:spoke-vn2	12	0	import_target target:1:2 target:64512:8000013	export_target target:1:1 target:64512:8000013
default-domain:admin:spoke-vn3:spoke-vn3	default-domain:admin:spoke-vn3	14	0	import_target target:1:2 target:64512:8000015	export_target target:1:1 target:64512:8000015
default-domain:admin:spoke-vn4:spoke-vn4	default-domain:admin:spoke-vn4	13	0	import_target target:1:2 target:64512:8000014	export_target target:1:1 target:64512:8000014

Remote Compute

IN THIS SECTION

Contrail Networking supports remote compute, a method of managing a Contrail deployment across many small distributed data centers efficiently and cost effectively.

Remote Compute Overview

Remote compute enables the deployment of Contrail Networking in many small distributed data centers, up to hundreds or even thousands, for telecommunications point-of-presence (PoPs) or central offices (COs). Each small data center has only a small number of computes, typically 5-20 in a rack, running a few applications such as video caching, traffic optimization, and virtual Broadband Network Gateway (vBNG). It is not cost effective to deploy a full Contrail controller cluster of nodes of control, configuration, analytics, database, and the like, in each distributed PoP on dedicated servers. Additionally, manually managing hundreds or thousands of clusters is not feasible operationally.

Remote Compute Features

Remote compute is implemented by means of a subcluster that manages compute nodes at remote sites to receive configurations and exchange routes.

The key concepts of Contrail remote compute include:

- Remote compute employs a subcluster to manage remote compute nodes away from the primary data center.
- The Contrail control cluster is deployed in large centralized data centers, where it can remotely manage compute nodes in small distributed small data centers.
- A lightweight version of the controller is created, limited to the control node, and the config node, analytics, and analytics database are shared across several control nodes.
- Many lightweight controllers are co-located on a small number of servers to optimize efficiency and cost.

- The control nodes peer with the remote compute nodes by means of XMPP and peer with local gateways by means of MP-eBGP.

Remote Compute Operations

A subcluster object is created for each remote site, with a list of links to local compute nodes that are represented as vrouter objects, and a list of links to local control nodes that are represented as BGP router objects, with an ASN as property.

The subclusters are identified in the provision script. The vrouter and bgp-router provision scripts take each subcluster as an optional argument to link or delink with the subcluster object.

It is recommended to spawn the control nodes of the remote cluster in the primary cluster, and they are IGBP-meshed among themselves within that subcluster. The control nodes BGP-peer with their respective SDN gateway, over which route exchange occurs with the primary control nodes.

Compute nodes in the remote site are provisioned to connect to their respective control nodes to receive configuration and exchange routes. Data communication among workloads between these clusters occurs through the provider backbone and their respective SDN gateways. The compute nodes and the control nodes push analytics data to analytics nodes hosted on the primary cluster.

Subcluster Properties

The Contrail Web UI shows a list of subcluster objects, each with a list of associated vrouters and BGP routers that are local in that remote site and the ASN property.

General properties of subclusters include:

- A subcluster control node never directly peers with another subcluster control node or with primary control nodes.
- A subcluster control node has to be created, and is referred to, in virtual-router and bgp-router objects.
- A subcluster object and the control nodes under it should have the same ASN.
- The ASN cannot be modified in a subcluster object.

NOTE: Multinode service chaining across subclusters is not supported.

Inter Subcluster Route Filtering

Contrail Networking Release 2005 supports inter subcluster route filtering (Beta). With this release, a new extended community called `origin-sub-cluster` (similar to `origin-vn`) is added to all routes originating from a subcluster.

The format of this new extended community is `subcluster:<`*asn*`>:<`*id*`>`.

This new extended community is added by encoding the subcluster ID in the ID field within the extended community. The subcluster ID helps you determine the subcluster from which the route originated, and is unique for each subcluster. For a 2-byte ASN format, type/subtype is 0x8085 and subcluster ID can be 4-byte long. For a 4-byte ASN format, type/subtype is 0x8285 and subcluster ID can be 2-byte long.

You create a routing policy matching this new extended community to be able to filter routes. Routing policies are always applied to primary routes. However, a routing policy is applied to a secondary route in the following scenarios:

- There is no subcluster extended community associated with the route.
- Self subcluster ID does not match the subcluster ID associated with the route.

Figure 5 on page 33 shows a data center network topology. All routing policies are configured on virtual networks in the main data center, POP0. Consider the following example routing policy:

```
From 0/0 & subcluster:<asn>:1 then LP=150
From 0/0 & subcluster:<asn>:2  then LP=140
From 0/0 then reject
```

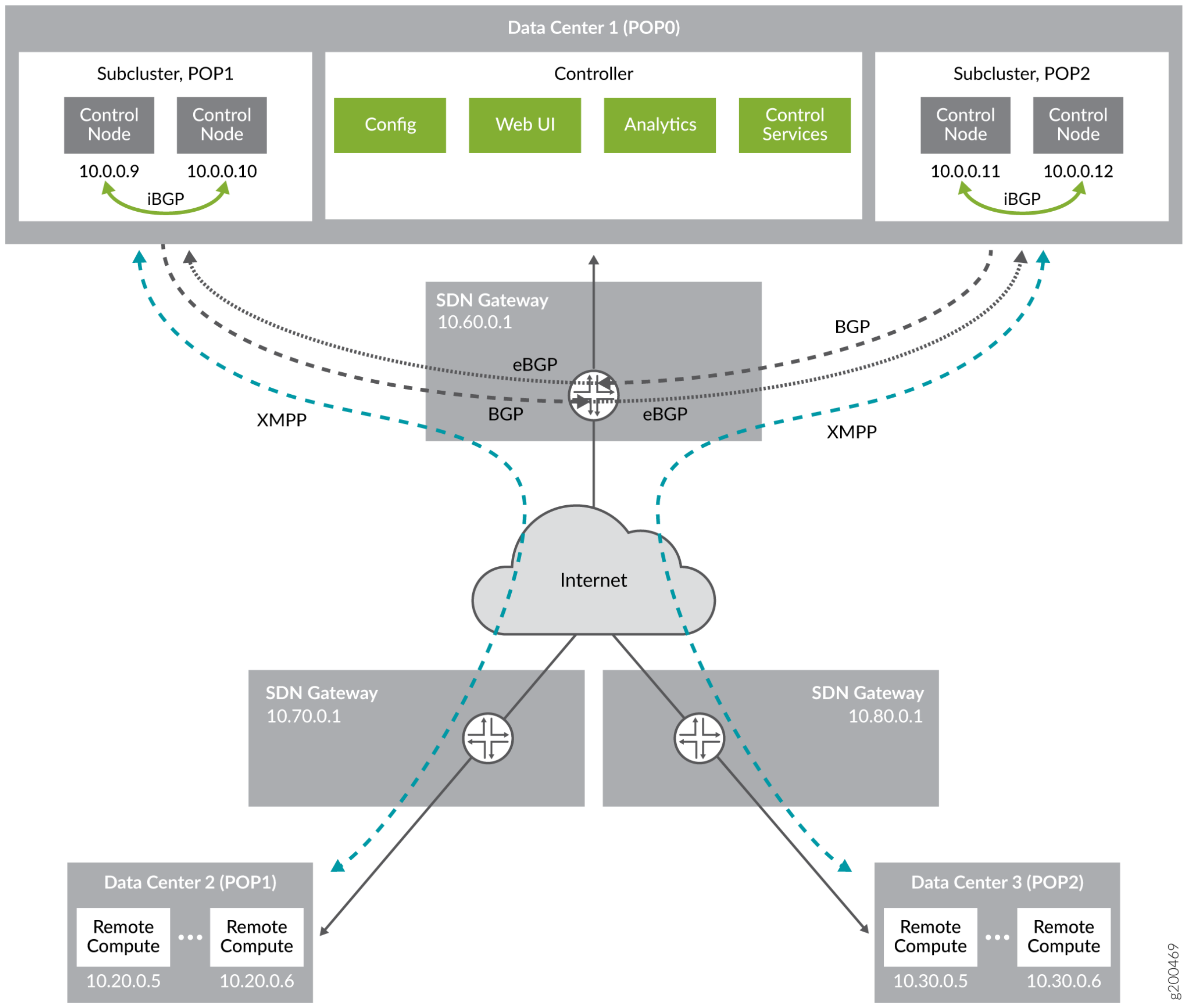
Where, `1` and `2` are the subcluster IDs of subclusters POP1 and POP2 respectively.

In this example, for routes directed to POP0 from subclusters POP1 and POP2, the LP will be changed. Routes that do not match the extended community are rejected. Default routes with no extended community are also rejected.

Provisioning a Remote Compute Cluster

Contrail Networking enables you to provision remote compute using an **instances.yaml** file. *Installing a Contrail Cluster using Contrail Command and instances.yml* shows a bare minimum configuration. The YAML file described in this section builds upon that minimum configuration and uses Figure 5 on page 33 as an example data center network topology.

Figure 5: Example Multi-Cluster Topology



In this topology, there is one main data center (**pop0**) and two remote data centers (**pop1** and **pop2**.) **pop0** contains two subclusters: one for **pop1,** and the other for **pop2**. Each subcluster has two control nodes. The control nodes within a subcluster, for example 10.0.0.9 and 10.0.0.10, communicate with each other through iBGP.

Communication between the control nodes within a subcluster and the remote data center is through the SDN Gateway; there is no direct connection. For example, the remote compute in pop1 (IP address 10.20.0.5) communicates with the control nodes (IP addresses 10.0.0.9 and 10.0.0.10) in subcluster 1 through the SDN Gateway.

To configure remote compute in the YAML file:

1. First, create the remote locations or subclusters. In this example, we create data centers 2 and 3 (with the names **pop1** and **pop2**, respectively), and define unique ASN numbers for each. Subcluster names must also be unique.

   ```
   remote_locations:
     pop1:
       BGP_ASN: 12345
       SUBCLUSTER: pop1
     pop2:
       BGP_ASN: 12346
       SUBCLUSTER: pop2
   ```

2. Create the control nodes for pop1 and pop2 and assign an IP address and role. These IP addresses are the local IP address. In this example, there are two control nodes for each subcluster.

   ```
   control_1_only_pop1:           # Mandatory. Instance name
       provider: bms              # Mandatory. Instance runs on BMS
       ip: 10.0.0.9
       roles:
         control:
           location: pop1
     control_2_only_pop1:         # Mandatory. Instance name
       provider: bms              # Mandatory. Instance runs on BMS
       ip: 10.0.0.10
       roles:
         control:
           location: pop1
     control_1_only_pop2:         # Mandatory. Instance name
       provider: bms              # Mandatory. Instance runs on BMS
       ip: 10.0.0.11
       roles:                     # Optional.
         control:
           location: pop2
     control_2_only_pop2:         # Mandatory. Instance name
       provider: bms              # Mandatory. Instance runs on BMS
       ip: 10.0.0.12
       roles:                     # Optional.
         control:
           location: pop2
   ```

3. Now, create the remote compute nodes for **pop1** and **pop2** and assign an IP address and role. In this example, there are two remote compute nodes for each data center. The 10.60.0.x addresses are the management IP addresses for the control service.

```
compute_1_pop1:                 # Mandatory. Instance name
    provider: bms               # Mandatory. Instance runs on BMS
    ip: 10.20.0.5
    roles:
      openstack_compute:        # Optional.
      vrouter:
        CONTROL_NODES: 10.60.0.9,10.60.0.10
        VROUTER_GATEWAY: 10.70.0.1
        location: pop1
  compute_2_pop1:               # Mandatory. Instance name
    provider: bms               # Mandatory. Instance runs on BMS
    ip: 10.20.0.6
    roles:
      openstack_compute:        # Optional.
      vrouter:
        CONTROL_NODES: 10.60.0.9,10.60.0.10
        VROUTER_GATEWAY: 10.70.0.1
        location: pop1
  compute_1_pop2:               # Mandatory. Instance name
    provider: bms               # Mandatory. Instance runs on BMS
    ip: 10.30.0.5
    roles:
      openstack_compute:        # Optional.
      vrouter:
        CONTROL_NODES: 10.60.0.11,10.60.0.12
        VROUTER_GATEWAY: 10.80.0.1
        location: pop2
  compute_2_pop2:               # Mandatory. Instance name
    provider: bms               # Mandatory. Instance runs on BMS
    ip: 10.30.0.6
    roles:
      openstack_compute:        # Optional.
      vrouter:
        CONTROL_NODES: 10.60.0.11,10.60.0.12
        VROUTER_GATEWAY: 10.80.0.1
        location: pop2
```

The entire YAML file is contained below.

Example instance.yaml with subcluster configuration

```
provider_config:
  bms:
    ssh_pwd: <password>
    ssh_user: <root_user>
    ntpserver: 10.84.5.100
    domainsuffix: local
instances:
  openstack_node:                  # Mandatory. Instance name
    provider: bms                  # Mandatory. Instance runs on BMS
    ip: 10.0.0.4
    roles:                         # Optional.
      openstack:
  all_contrail_roles_default_pop:  # Mandatory. Instance name
    provider: bms                  # Mandatory. Instance runs on BMS
    ip: 10.0.0.5
    roles:                         # Optional.
      config_database:             # Optional.
      config:                      # Optional.
      control:                     # Optional.
      analytics_database:          # Optional.
      analytics:                   # Optional.
      webui:                       # Optional.
  compute_3_default_pop:           # Mandatory. Instance name
    provider: bms                  # Mandatory. Instance runs on BMS
    ip: 10.0.0.6
    roles:
      openstack_compute:
      vrouter:
        VROUTER_GATEWAY: 10.60.0.1
  compute_1_default_pop:           # Mandatory. Instance name
    provider: bms                  # Mandatory. Instance runs on BMS
    ip: 10.0.0.7
    roles:
      openstack_compute:
      vrouter:
        VROUTER_GATEWAY: 10.60.0.1
  compute_2_default_pop:          # Mandatory. Instance name
    provider: bms                 # Mandatory. Instance runs on BMS
    ip: 10.0.0.8
    roles:
```

```
    openstack_compute:
    vrouter:
      VROUTER_GATEWAY: 10.60.0.1
control_1_only_pop1:            # Mandatory. Instance name
  provider: bms                 # Mandatory. Instance runs on BMS
  ip: 10.0.0.9
  roles:
    control:
      location: pop1
control_2_only_pop1:            # Mandatory. Instance name
  provider: bms                 # Mandatory. Instance runs on BMS
  ip: 10.0.0.10
  roles:
    control:
      location: pop1
control_1_only_pop2:            # Mandatory. Instance name
  provider: bms                 # Mandatory. Instance runs on BMS
  ip: 10.0.0.11
  roles:                        # Optional.
    control:
      location: pop2
control_2_only_pop2:            # Mandatory. Instance name
  provider: bms                 # Mandatory. Instance runs on BMS
  ip: 10.0.0.12
  roles:                        # Optional.
    control:
      location: pop2
compute_1_pop1:                 # Mandatory. Instance name
  provider: bms                 # Mandatory. Instance runs on BMS
  ip: 10.20.0.5
  roles:
    openstack_compute:          # Optional.
    vrouter:
      CONTROL_NODES: 10.60.0.9,10.60.0.10
      VROUTER_GATEWAY: 10.70.0.1
      location: pop1
compute_2_pop1:                 # Mandatory. Instance name
  provider: bms                 # Mandatory. Instance runs on BMS
  ip: 10.20.0.6
  roles:
    openstack_compute:          # Optional.
    vrouter:
      CONTROL_NODES: 10.60.0.9,10.60.0.10
```

```
        VROUTER_GATEWAY: 10.70.0.1
        location: pop1
  compute_1_pop2:                 # Mandatory. Instance name
    provider: bms                 # Mandatory. Instance runs on BMS
    ip: 10.30.0.5
    roles:
      openstack_compute:          # Optional.
      vrouter:
        CONTROL_NODES: 10.60.0.11,10.60.0.12
        VROUTER_GATEWAY: 10.80.0.1
        location: pop2
  compute_2_pop2:                 # Mandatory. Instance name
    provider: bms                 # Mandatory. Instance runs on BMS
    ip: 10.30.0.6
    roles:
      openstack_compute:          # Optional.
      vrouter:
        CONTROL_NODES: 10.60.0.11,10.60.0.12
        VROUTER_GATEWAY: 10.80.0.1
        location: pop2
global_configuration:
  CONTAINER_REGISTRY: 10.xx.x.81:5000
  REGISTRY_PRIVATE_INSECURE: True

contrail_configuration:           # Contrail service configuration section
  CONTRAIL_VERSION: <contrail_version>
  CONTROLLER_NODES: 10.60.0.5
  CLOUD_ORCHESTRATOR: openstack
  KEYSTONE_AUTH_HOST: 10.60.0.100
  KEYSTONE_AUTH_URL_VERSION: /v3
  RABBITMQ_NODE_PORT: 5673
  PHYSICAL_INTERFACE: eth1
  CONTROL_DATA_NET_LIST: 10.60.0.0/24,10.70.0.0/24,10.80.0.0/24

kolla_config:
  kolla_globals:
    network_interface: "eth1"
    enable_haproxy: "yes"
    contrail_api_interface_address: 10.60.0.5
    kolla_internal_vip_address: 10.60.0.100
    kolla_external_vip_address: 10.0.0.100
    kolla_external_vip_interface: "eth0"
  kolla_passwords:
    keystone_admin_password: <password>

remote_locations:
  pop1:
    BGP_ASN: 12345
    SUBCLUSTER: pop1
  pop2:
    BGP_ASN: 12346
    SUBCLUSTER: pop2
```

NOTE: Replace *<contrail_version>* with the correct `contrail_container_tag` value for your Contrail Networking release. The respective `contrail_container_tag` values are listed in README Access to Contrail Registry.

Automatically Deploy Remote Compute Using RHOSP/TripleO

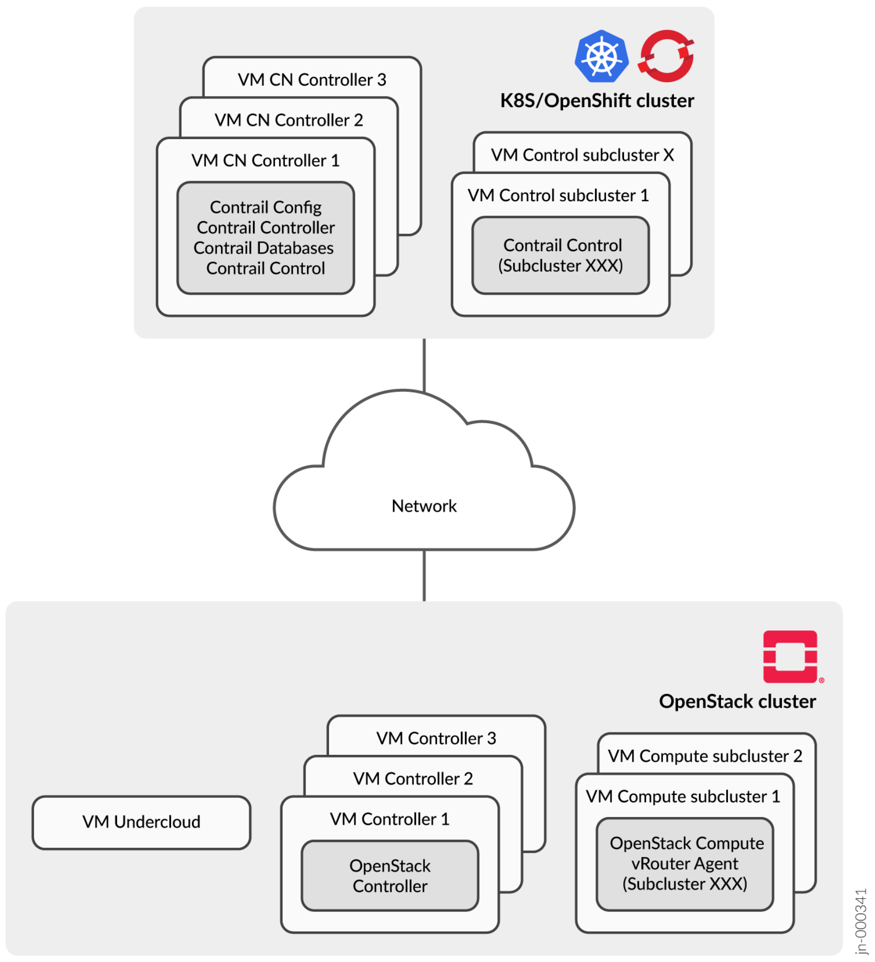
A Distributed compute node (DCN) architecture is designed for edge use cases, allowing remote compute and storage nodes to be deployed remotely while sharing a common centralized control plane. The DCN architecture allows you to strategically position workloads closer to your operational needs for higher performance.

Starting in Contrail Networking 21.4, you can deploy remote compute automatically using RHOSP/TripleO for edge use cases.

Example Topology

You can build the setup in different ways by providing the control plane elements. Figure 6 on page 40 shows the example setup for deploying the remote compute automatically.

Figure 6: Example Setup for Deploying the Remote Compute Automatically



In this example:

- Setup is explained without spine-leaf and/or the DCN details. See Figure 6 on page 40.
- Describe primarily on the Contrail specific configuration. All scripts provided are only examples. For deployment preparation instructions, see RedHat documentation.
- All control plane functions are provided as virtual machines hosted on the KVM hosts:
  - VM 1—Kubernetes managed: Contrail Control plane (Kubernetes master)
  - VM 2—Kubernetes managed: Contrail Control service for remote compute (non-master Kubernetes node with a subcluster label)
  - VM 3—RHOSP undercloud

    - VM 4—RHOSP overcloud: OpenStack Controller
    - VM 5—RHOSP overcloud: OpenStack remote compute with subcluster param
- Contrail control plane uses Kubernetes cluster. You can do the same with OpenShift.

Prepare Kubernetes Managed Hosts

To prepare Kubernetes managed hosts:

1. Create two Contrail master and Contrail controller machines with the following specifications:
    - CentOS 7
    - 32GB RAM
    - 80GB SDD
2. Deploy the Contrail Control plane in a Kubernetes cluster with at least one worker node using tf-operator.

    The worker will be used for Contrail Control serving a subcluster (one for testing and minimum two for production). In the case of OpenShift, see the Contrail Control plane's Readme file.

    If RHOSP uses TLS everywhere, you must deploy the Contrail Control plane with a CA bundle that includes both your own root CA and IPA CA data. For example:

```
# assuming that Kubernetes cluster ca is in ca.crt.pem and IPA CA is in ipa.crt
# (ipa.crt can be copied from undercloud node from /etc/ipa/ca.crt)
cat ca.crt.pem ipa.crt > ca-bundle.pem

# Assuming that Kubernetes cluster CA key is in ca.key.pem
export CERT_SIGNER="SelfSignedCA"
export TF_ROOT_CA_KEY_BASE64=$(cat ca.key.pem | base64 -w 0)
export TF_ROOT_CA_CERT_BASE64=$(cat ca-bundle.pem | base64 -w 0)

... other actions to deploy from tf-operator ...
```

3. Label worker(s) node with subcluster label.

```
# For each subcluster nodes
kubectl label node <worker_nodename> subcluster=<subcluster_name>
```

4. Ensure Kubernetes nodes can:

- Connect to external, internal API, and tenant RHOSP networks.
- Resolve RHOSP FQDNs for overcloud VIPs for external, internal API, and Control plane networks.

  You can obtain FQDNs of overcloud nodes from **/etc/hosts** of one of the overcloud node.

For example:

```
cat /etc/hosts

192.x.x.x overcloud.ctlplane.5c7.local
10.x.x.x overcloud.internalapi.5c7.local
10.x.x.x overcloud.5c7.local overcloud.5c7.local

#RHOSP Computes
192.x.x.x  overcloud-remotecompute1-0.tenant.dev.localdomain
# ...
#RHOSP Contrail Dpdk
192.x.x.x  overcloud-remotecontraildpdk1-0.tenant.dev.localdomain
# ...
#RHOSP Contrail Sriov
192.x.x.x  overcloud-remotecontrailsriov1-0.tenant.dev.localdomain
# ...
#... other compute addresses if any
... IMPORTANT: all FQDNs of all overcloud nodes (all networks) ...
```

5. Edit the manager manifest to add one more control with a node selector and a subcluster parameter.

```
kubectl edit manager -n tf
```

Add a record to each subcluster's controls:

```
    controls:
    - metadata:
        labels:
          tf_cluster: cluster1
        name: control<subcluster_name>
      spec:
        commonConfiguration:
          nodeSelector:
            subcluster: <subcluster_name>
        serviceConfiguration:
          subcluster: <subcluster_name>
          asnNumber: <asn>
          containers:
          - name: control
            image: contrail-controller-control-control
          - name: dns
            image: contrail-controller-control-dns
          - name: named
            image: contrail-controller-control-named
          - name: nodemanager
            image: contrail-nodemgr
          - name: provisioner
            image: contrail-provisioner
```

Prepare OpenStack Managed Hosts

To prepare OpenStack managed hosts:

1. Prepare OpenStack hosts and run undercloud setup.

2. Run the following script to generate remote computes heat templates for kernel, DPDK, and SR-IOV:

```
cd
# Comma separated list of names
subcluster_names=pop1,pop2
./tripleo-heat-templates/tools/contrail/remote_compute.sh $subcluster_names
```

   This script generates a **network_data_rcomp.yaml** file and the set of files for each subcluster. For example:

   - **tripleo-heat-templates/roles/RemoteCompute1.yaml**
   - **tripleo-heat-templates/roles/RemoteContrailDpdk1.yaml**
   - **tripleo-heat-templates/roles/RemoteContrailSriov1.yaml**
   - **tripleo-heat-templates/environments/contrail/rcomp1-env.yaml**
   - **tripleo-heat-templates/network/config/contrail/compute-nic-config-rcomp1.yaml**
   - **tripleo-heat-templates/network/config/contrail/contrail-dpdk-nic-config-rcomp1.yaml**
   - **tripleo-heat-templates/network/config/contrail/contrail-sriov-nic-config-rcomp1.yaml**

3. Ensure that the generated files and other templates are customized to your setup (storage, network CIDRs, and routes). For more information, see RedHat documentation.

4. Prepare Contrail templates using the generated network data file:

   a. Modify **contrail-services.yaml** to provide data about the Contrail Control plane on Kubernetes.

```
  # Set keystone admin port to be on internal_api
  ServiceNetMap:
    # ... others options...
    KeystoneAdminApiNetwork: internal_api

  # FQDN resolving
  ExtraHostFileEntries:
    - 'IP1    <FQDN K8S master1>        <Short name master1>'
    - 'IP2    <FQDN K8S master2>        <Short name master2>'
    - 'IP3    <FQDN K8S master3>        <Short name master3>'
    - 'IP4    <FQDN K8S pop1 worker1>   <Short name pop1 worker1>'
    - 'IP5    <FQDN K8S pop1 worker2>   <Short name pop1 worker2>'
    - 'IP6    <FQDN K8S pop2 worker1>   <Short name pop2 worker1>'
    - 'IP7    <FQDN K8S pop2 worker2>   <Short name pop2 worker2>'

  # Main control plane
  ExternalContrailConfigIPs: <comma separated list of IP/FQDNs of K8S master nodes>
  ExternalContrailControlIPs: <comma separated list of IP/FQDNs of K8S master nodes>
  ExternalContrailAnalyticsIPs: <comma separated list of IP/FQDNs of K8S master nodes>

  ControllerExtraConfig:
    contrail_internal_api_ssl: True
  ComputeExtraConfig:
    contrail_internal_api_ssl: True
  # Add contrail_internal_api_ssl for all other roles if any
```

   b. Enable Contrail TLS 4.2.1 if RHOSP does not use TLS everywhere or use self-signed root CA.

   Prepare self-signed certificates in **environments/contrail/contrail-tls.yaml**.

```
resource_registry:
  OS::TripleO::Services::ContrailCertmongerUser: OS::Heat::None

parameter_defaults:
  ContrailSslEnabled: true
  ContrailServiceCertFile: '/etc/contrail/ssl/certs/server.pem'
  ContrailServiceKeyFile: '/etc/contrail/ssl/private/server-privkey.pem'
  ContrailCA: 'local'
  ContrailCaCertFile: '/etc/contrail/ssl/certs/ca-cert.pem'
  ContrailCaKeyFile: '/etc/contrail/ssl/private/ca-key.pem'
  ContrailCaCert: |
    <Root CA certificate from K8S setup>>
  ContrailCaKey: |
    <Root CA private key from K8S setup>
```

If RHOSP uses TLS everywhere, do the following:

i. Make a CA bundle file.

```
# Assuming that k8s cluster ca is in ca.crt.pem
cat /etc/ipa/ca.crt ca.crt.pem > ca-bundle.pem
```

ii. Prepare an environment file **ca-bundle.yaml**.

```
# Create file
cat <<EOF > ca-bundle.yaml
resource_registry:
  OS::TripleO::NodeTLSCAData: tripleo-heat-templates/puppet/extraconfig/tls/ca-
inject.yaml
parameter_defaults:
  ContrailCaCertFile: "/etc/pki/ca-trust/source/anchors/contrail-ca-cert.pem"
  SSLRootCertificatePath: "/etc/pki/ca-trust/source/anchors/contrail-ca-cert.pem"
  SSLRootCertificate: |
EOF
# Append cert data
cat ca-bundle.pem | while read l ; do
  echo "    $l" >> ca-bundle.yaml
done
# Check
cat ca-bundle.yaml
```

c. Prepare central site-specific parameters.

```
# !!! IMPORTANTN: Adjust to your setup
# (Check more options in RedHat doc)
```

```
cat <<EOF > central-env.yaml
parameter_defaults:
  GlanceBackend: swift
  ManageNetworks: true
  ControlPlaneSubnet: leaf0
  ControlControlPlaneSubnet: leaf0
  InternalApiInterfaceRoutes:
    - destination: 10.x.x.x/24
      nexthop: 10.x.x.x
    - destination: 10.x.x.x/24
      nexthop: 10.x.x.x
  StorageMgmtInterfaceRoutes:
    - destination: 10.x.x.x/24
      nexthop: 10.x.x.x
    - destination: 10.x.x.x/24
      nexthop: 10.x.x.x
  StorageInterfaceRoutes:
    - destination: 10.x.x.x/24
      nexthop: 10.x.x.x
    - destination: 10.x.x.x/24
      nexthop: 10.x.x.x
  TenantInterfaceRoutes:
    - destination: 172.x.x.x/24
      nexthop: 172.x.x.x
  ControlPlaneStaticRoutes:
    - destination: 172.x.x.x/24
      nexthop: 192.x.x.x
    - destination: 172.x.x.x/24
      nexthop: 192.x.x.x
  NovaComputeAvailabilityZone: 'central'
  ControllerExtraConfig:
    nova::availability_zone::default_schedule_zone: central
  NovaCrossAZAttach: false
  CinderStorageAvailabilityZone: 'central'
EOF

# If use tenant network on openstack controllers adjust nic file. For example:
# vi tripleo-heat-templates/network/config/contrail/controller-nic-config.yaml
              - type: interface
                name: nic2
                use_dhcp: false
                addresses:
                - ip_netmask:
                    get_param: TenantIpSubnet
                  routes:
                    get_param: TenantInterfaceRoutes
```

d. Prepare VIP mapping.

```
# !!! Adjust to your setup
# Check more options in RedHat doc
cat <<EOF > leaf-vips.yaml
parameter_defaults:
  VipSubnetMap:
    ctlplane: leaf0
    redis: internal_api_subnet
    InternalApi: internal_api_subnet
    Storage: storage_subnet
    StorageMgmt: storage_mgmt_subnet
EOF
```

e. Generate role and network files using heat templates.

```
cd
# generate role file (adjust to your role list)
openstack overcloud roles generate --roles-path tripleo-heat-templates/roles \
  -o /home/stack/roles_data.yaml Controller RemoteCompute1
# clean old files if any
./tripleo-heat-templates/tools/process-templates.py --clean \
  -r /home/stack/roles_data.yaml \
  -n /home/stack/tripleo-heat-templates/network_data_rcomp.yaml \
  -p tripleo-heat-templates/
# generated tripleo stack files
./tripleo-heat-templates/tools/process-templates.py \
  -r /home/stack/roles_data.yaml \
  -n /home/stack/tripleo-heat-templates/network_data_rcomp.yaml \
  -p tripleo-heat-templates/
```

5. Deploy the central location.

```
# Example for the case when RHOSP uses TLS everywhere
# use generated role file, network data file and files for remote computes
openstack overcloud deploy --templates tripleo-heat-templates/ \
  --stack overcloud --libvirt-type kvm \
  --roles-file /home/stack/roles_data.yaml \
  -n /home/stack/tripleo-heat-templates/network_data_rcomp.yaml \
  -e tripleo-heat-templates/environments/rhsm.yaml \
  -e tripleo-heat-templates/environments/network-isolation.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-services.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-net.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-plugins.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-tls.yaml \
  -e tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml \
  -e tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
  -e containers-prepare-parameter.yaml \
  -e rhsm.yaml \
  -e ca-bundle.yaml \
  -e central-env.yaml \
  -e leaf-vips.yaml
```

6. Enable keystone authentication for the Kubernetes cluster, if it is not already enabled.

```
# Ensure that all Kubernetes nodes can resolve overcloud VIPs FQDNs like
overcloud.internalapi.5c7.local
[stack@node1 ~]$ grep overcloud.internalapi.5c7.local  /etc/hosts
10.1.0.125 overcloud.internalapi.5c7.local
...

# Edit manager object to put keystone parameters and set linklocal parameters
kubectl -n tf edit managers cluster1

# Example of configuration
apiVersion: tf.tungsten.io/v1alpha1
kind: Manager
metadata:
  name: cluster1
  namespace: tf
spec:
  commonConfiguration:
    authParameters:
      authMode: keystone
      keystoneAuthParameters:
        address: overcloud.internalapi.5c7.local
        adminPassword: <password>
```

```
          authProtocol: https
          region: regionOne
...
    config:
      metadata:
        labels:
          tf_cluster: cluster1
        name: config1
      spec:
        commonConfiguration:
          nodeSelector:
            node-role.kubernetes.io/master: ""
        serviceConfiguration:
          linklocalServiceConfig:
            ipFabricServiceHost: "overcloud.internalapi.5c7.local"
...
```

7. Deploy the remote sites. For example, export environment from the central site.

```
mkdir -p ~/dcn-common
openstack overcloud export \
  --stack overcloud \
  --config-download-dir /var/lib/mistral/overcloud \
  --output-file ~/dcn-common/central-export.yaml
```

   a. Deploy the remote site 1.

```
openstack overcloud deploy --templates tripleo-heat-templates/ \
  --stack pop1 --libvirt-type kvm \
  --roles-file /home/stack/roles_data.yaml \
  -n /home/stack/tripleo-heat-templates/network_data_rcomp.yaml \
  -e tripleo-heat-templates/environments/rhsm.yaml \
  -e tripleo-heat-templates/environments/network-isolation.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-services.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-net.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-plugins.yaml \
  -e tripleo-heat-templates/environments/contrail/contrail-tls.yaml \
  -e tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-dns.yaml \
  -e tripleo-heat-templates/environments/services/haproxy-public-tls-certmonger.yaml \
  -e tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
  -e containers-prepare-parameter.yaml \
  -e rhsm.yaml \
  -e ca-bundle.yaml \
  -e dcn-common/central-export.yaml \
  -e leaf-vips.yaml \
  -e /home/stack/tripleo-heat-templates/environments/contrail/rcomp1-env.yaml
```

b. Follow the next steps from RedHat documentation.

8. Deploy the edge sites with storage.

   Ensure that Nova cell_v2 host mappings are created in the Nova API database after the edge locations are deployed.

   Run the following command on the undercloud:

```
TRIPLEO_PLAN_NAME=overcloud \
  ansible -i /usr/bin/tripleo-ansible-inventory \
    nova_api[0] -b -a \
    "{{ container_cli }} exec -it nova_api \
      nova-manage cell_v2 discover_hosts --by-service --verbose"
```

Monitoring Remote Compute Clusters in Contrail Command

Starting in Contrail Networking Release 21.4.L1, you can use the Contrail Command graphical user interface (GUI) to monitor remote compute routing clusters.

You can gather the following data in an easy-to-read graphical presentation about any remote compute routing cluster in your environment:

- Nodes
- Updates Sent per Node
- Updates Received per Node
- BGP CPU Share per Node
- BGP Memory Usage per Node

To view monitoring data about remote compute routing clusters, navigate to the **Monitoring** > **Dashboards** > **Routing Cluster** tab in Contrail Command.

Monitoring and Configuring BGP Routers for Remote Compute in Contrail Command

Starting in Contrail Networking Release 21.4.L1, enhancements were made to the Contrail Command graphical user interface (GUI) that allowed users to better monitor and configure BGP Routers with remote compute routing clusters..

These enhancements included:

- On the **Create BGP Router** page in the **Infrastructure** > **Cluster** > **Advanced** > **BGP Routers** page, you can now connect a remote routing cluster to a BGP Router.

  This option is now available when you navigate to the **Advanced Options** > **Routing Cluster Id** drop-down menu. You can select a remote compute routing cluster, which will be available as an option in the **Routing Cluster Id** drop-down menu, from the page.

- On the **BGP Routers** tab in the **Infrastructure** > **Cluster** > **Advanced** page, you can now view the BGP router connections to the remote compute routing clusters.

You, notably, cannot edit or change a router cluster from configured BGP routers within Contrail Command. You cannot, for instance, edit or change the router cluster from the router cluster drop down menu on the **Infrastructures** > **Cluster** > **Advanced Options** > **BGP Router** > **Create** > **Advanced Options** > **Routing Cluster-ID** > **Routing Cluster ASN** page.

Viewing the Virtual Router Connected to Remote Compute Clusters in Contrail Command

Starting in Contrail Networking Release 21.4.L1, you can view the virtual router connected to a remote compute routing cluster in Contrail Command. The Virtual Routers will appear in the table on the **Infrastructure** > **Cluster** > **Advanced** > **Virtual Routers** page

BGP as a Service (BGPaaS) Support in Remote Compute Clusters

Starting in Contrail Networking Release 21.4.L2, you can configure BGP as a Service (BGPaaS) in remote compute clusters.

You configure BGPaaS in a remote compute cluster in the same manner that you would configure outside of a remote compute cluster. For information on configuring BGPaaS in Contrail Networking, see BGP as a Service.

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
2005	Contrail Networking Release 2005 supports inter subcluster route filtering (Beta).
21.4.L2	Starting in Contrail Networking Release 21.4.L2, you can configure BGP as a Service (BGPaaS) in remote compute clusters.
21.4.L1	Starting in Contrail Networking Release 21.4.L1, you can use the Contrail Command graphical user interface (GUI) to monitor remote compute routing clusters.
21.4.L1	Starting in Contrail Networking Release 21.4.L1, enhancements were made to the Contrail Command graphical user interface (GUI) that allowed users to better monitor and configure BGP Routers with remote compute routing clusters.
21.4.L1	Starting in Contrail Networking Release 21.4.L1, you can view the virtual router connected to a remote compute routing cluster in Contrail Command.
21.4	Starting in Contrail Networking 21.4, you can deploy remote compute automatically using RHOSP/ TripleO for edge use cases.

3

CHAPTER

Advanced Service Chain Configuration

Customized Hash Field Selection for ECMP Load Balancing

IN THIS SECTION

Overview: Custom Hash Feature

Contrail Networking enables you to configure the set of fields used to hash upon during equal-cost multipath (ECMP) load balancing.

With the custom hash feature, users can configure an exact subset of fields to hash upon when choosing the forwarding path among a set of eligible ECMP candidates.

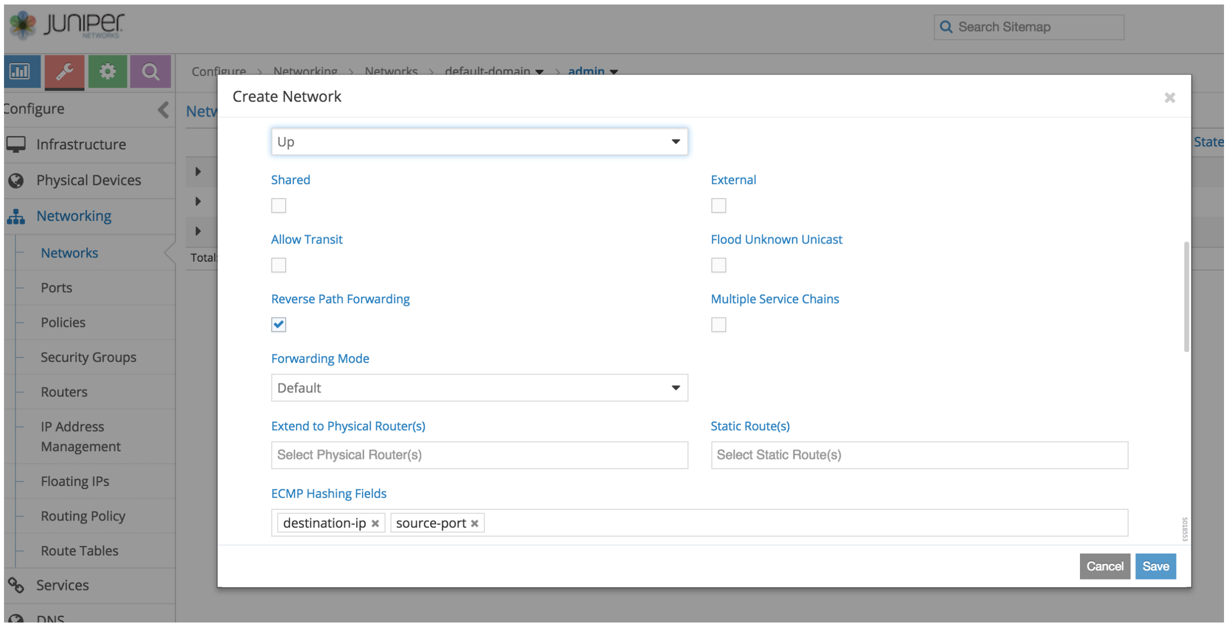
The custom hash configuration can be applied in the following ways:

- globally
- per virtual network (VN)
- per virtual network interface (VMI)

VMI configurations take precedence over VN configurations, and VN configurations take precedence over global level configuration (if present).

Custom hash is useful whenever packets originating from a particular source and addressed to a particular destination must go through the same set of service instances during transit. This might be required if source, destination, or transit nodes maintain a certain state based on the flow, and the state behavior could also be used for subsequent new flows, between the same pair of source and destination addresses. In such cases, subsequent flows must follow the same set of service nodes followed by the initial flow.

You can use the Contrail Web UI to identify specific fields in the network upon which to hash at the **Configure > Networking > Network, Create Network** window, in the **ECMP Hashing Fields** section as shown in the following figure.



Create Network

Admin State: Up

Shared: ☐ External: ☐

Allow Transit: ☐ Flood Unknown Unicast: ☐

Reverse Path Forwarding: ✓ Multiple Service Chains: ☐

Forwarding Mode: Default

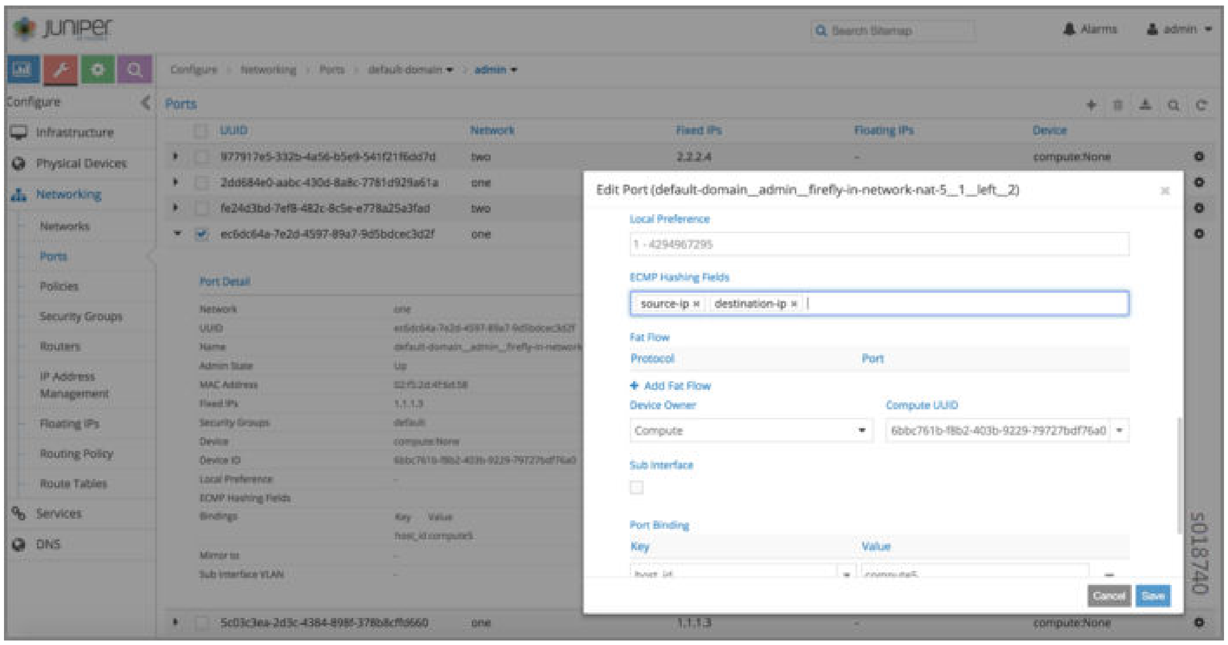
Extend to Physical Router(s): Select Physical Router(s)

Static Route(s): Select Static Route(s)

ECMP Hashing Fields: destination-ip × source-port ×

If the hashing fields are configured for a virtual network, all traffic destined to that VN will be subject to the customized hash field selection during forwarding over ECMP paths by vRouters. This may not be desirable in all cases, as it could potentially skew all traffic to the destination network over a smaller set of paths across the IP fabric.

A more practical scenario is one in which flows between a source and destination must go through the same service instance in between, where one could configure customized ECMP fields for the virtual machine interface of the service instance. Then, each service chain route originating from that virtual machine interface would get the desired ECMP field selection applied as its path attribute, and eventually get propagated to the ingress vRouter node. See the following example.



UUID	Network	Fixed IPs	Floating IPs	Device
977917e5-332b-4a56-b5e9-541f21f6dd7d	two	2.2.2.4	-	compute:None
2dd684e0-aabc-430d-8a8c-7781d929a61a	one			
fe24d3bd-7ef8-482c-8c5e-e778a25a3fad	two			
ec6dc64a-7e2d-4597-89a7-9d5bdcec3d2f	one			
5c03c3ea-2d3c-4384-898f-378b8cffd660	one	1.1.1.3	-	compute:None

Edit Port (default-domain__admin__firefly-in-network-nat-5__1__left__2)

Local Preference: 1 - 4294967295

ECMP Hashing Fields: source-ip × destination-ip ×

Fat Flow: Protocol Port

Device Owner: Compute

Compute UUID: 6bbc761b-f8b2-403b-9229-79727bdf76a0

Sub Interface: ☐

Using ECMP Hash Fields Selection

Custom hash fields selection is most useful in scenarios where multiple ECMP paths exist for a destination. Typically, the multiple ECMP paths point to ingress service instance nodes, which could be running anywhere in the Contrail cloud.

Configuring ECMP Hash Fields Over Service Chains

Use the following steps to create customized hash fields with ECMP over service chains.

1. Create the virtual networks needed to interconnect using service chaining, with ECMP load-balancing.

2. Create a service template and enable scaling.

3. Create a service instance, and using the service template, configure by selecting:

   - the desired number of instances for scale-out

   - the left and right virtual network to connect

   - the shared address space, to make sure that instantiated services come up with the same IP address for left and right, respectively

   This configuration enables ECMP among all those service instances during forwarding.

4. Create a policy, then select the service instance previously created and apply the policy to to the desired VMIs or VNs.

5. After the service VMs are instantiated, the ports of the left and right interfaces are available for further configuration. At the Contrail Web UI Ports section under Networking, select the ports on the left interface (virtual machine interface) of the service instance and apply the desired ECMP hash field configuration.

   NOTE: Currently the ECMP field selection configuration for the service instance left or right interface must be applied by using the Ports (VMIs) section under Networking and explicitly configuring the ECMP fields selection for each of the instantiated service instances' VMIs. This must be done for all service interfaces of the group, to ensure the end result is as expected, because the load balance attribute of only the best path is carried over to the ingress vRouter. If the load balance attribute is not configured, it is not propagated to the ingress vRouter, even if other paths have that configuration.

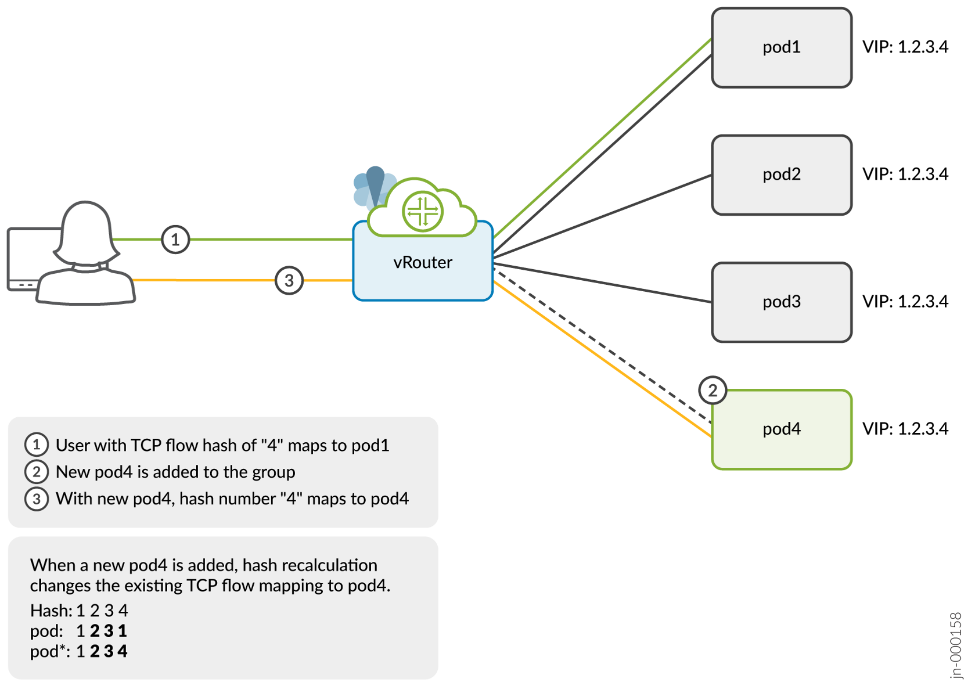
When the configuration is finished, the vRouters get programmed with routing tables with the ECMP paths to the various service instances. The vRouters are also programmed with the desired ECMP hash fields to be used during load balancing of the traffic.

Flow Stickiness for Load-Balanced System

Flow stickiness is a beta feature in Contrail Networking Release 21.3 that helps to minimize flow remapping across equal cost multipath (ECMP) groups in a load-balanced system.

We'll show you an example of a flow remapping problem that occurs when a new member is added to a three-member ECMP load-balancing system. See Figure 7 on page 57 for the workflow.

Figure 7: Example for Scale-Up Scenario


pod1
VIP: 1.2.3.4
pod2
VIP: 1.2.3.4
pod3
VIP: 1.2.3.4
pod4
VIP: 1.2.3.4
vRouter
1
2
3
1 User with TCP flow hash of "4" maps to pod1
2 New pod4 is added to the group
3 With new pod4, hash number "4" maps to pod4
When a new pod4 is added, hash recalculation changes the existing TCP flow mapping to pod4.
Hash: 1 2 3 4
pod: 1 2 3 1
pod*: 1 2 3 4
jn-000158

In this example, you'll send a flow request to the IP address 1.2.3.4. Because this is a three members group with the VIP address 1.2.3.4, vRouter sends the request to pod1 based on the flow hash calculation. Let's add a new member pod4 to the same VIP group. The request is now diverted and redirected to pod4 based on the flow hash recalculation.

Flow stickiness reduces such flow being remapped and retains the flow with the original path to pod1 though a new member pod4 is added. If adding pod4 affects the flow, vRouter reprograms the flow table and rebalances the flow with pod1.

Table 3 on page 58 shows the expected normal hash and flow stickiness results for the scale-up and scale-down scenarios.

Table 3: Expected Results When Members Are Added to or Deleted from ECMP Group

Example Scenario	Normal (Static) Hash Result	Flow Stickiness Result
ECMP group size is 3.	Based on the flow hash, traffic will be directed to pod1.	Based on the flow hash, traffic will be directed to pod1.
Add one more pod to the same service. ECMP group size is 4.	Flow redistribution is possible and traffic may now be redirected to another pod.	Traffic will continue to be directed to pod1.
Delete one pod from same service. ECMP group size is 2.	Flow redistribution is possible and traffic may now be redirected to another pod.	Unless pod1 is deleted, traffic will continue to be directed to it. If pod1 is deleted, the session must be reinitiated from client.

Flow stickiness is only supported when the flow is an ECMP flow before and after scaling up or scaling down.

Here’s an example of how flow stickiness may not work as expected:

Example Deployment	Scenario	Result
Two pods in two computes, one in each.	Before scale-up	With respect to the compute forwarding the traffic, flow will be a non-ecmp.
	After scale-up	Flow becomes an ECMP flow and triggers rehashing. This may cause the flow stickiness to fail.

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
21.3	Flow stickiness is a beta feature in Contrail Networking Release 21.3 that helps to minimize flow remapping across equal cost multipath (ECMP) groups in a load-balanced system.

Routing Policy

IN THIS SECTION

Starting with Contrail Networking Release 1910, virtual network routing policies are automatically applied to secondary routes. See "Applying Routing Policies to Secondary Routes" on page 63.

Contrail Networking uses routing policy infrastructure to manipulate the route and path attribute dynamically and supports attaching the import routing policy on the service instances.

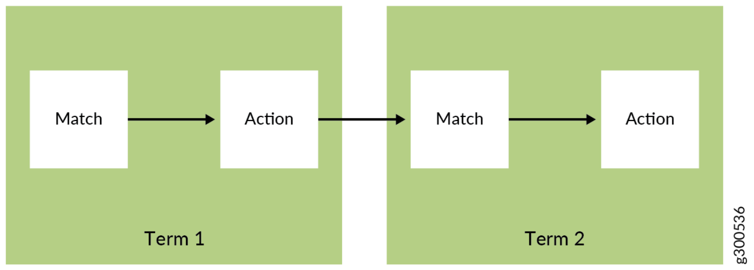
The routing policy contains list terms. A term can be a terminal rule, meaning that upon a match on the specified term, no further terms are evaluated and the route is dropped or accepted, based on the action in that term.

If the term is not a terminal rule, subsequent terms are evaluated for the given route.

The list terms are structured as in the following example.

```
Policy {
     Term-1
     Term-2
}
```

The matches and actions of the policy term lists operate similarly to the Junos language match and actions operations. A visual representation is the following.



Each term is represented as in the following:

```
from {
    match-condition-1
    match-condition-2
    ..
    ..
}
then {
    action
    update-action-1
    update-action-2
    ..
    ..
}
```

The term should not contain an `any` match condition, for example, an empty `from` should not be present.

If an `any` match condition is present, all routes are considered as matching the term.

However, the `then` condition can be empty or the action can be unspecified.

Applying Routing Policy

The routing policy evaluation has the following key points:

- If the term of a routing policy consists of multiple match conditions, a route must satisfy all match conditions to apply the action specified in the term.
- If a term in the policy does not specify a match condition, all routes are evaluated against the match.

- If a match occurs but the policy does not specify an accept, reject, or next term action, one of the following occurs:
  - The next term, if present, is evaluated.
  - If no other terms are present, the next policy is evaluated.
  - If no other policies are present, the route is accepted. The default routing policy action is “accept”.
- If a match does not occur with a term in a policy, and subsequent terms in the same policy exist, the next term is evaluated.
- If a match does not occur with any terms in a policy, and subsequent policies exist, the next policy is evaluated.
- If a match does not occur by the end of a policy or all policies, the route is accepted.

A routing policy can consist of multiple terms. Each term consists of match conditions and actions to apply to matching routes.

Each route is evaluated against the policy as follows:

1. The route is evaluated against the first term. If it matches, the specified action is taken. If the action is to accept or reject the route, that action is taken and the evaluation of the route ends. If the next term action is specified or if no action is specified, or if the route does not match, the evaluation continues as described above to subsequent terms.
2. Upon hitting the last non-terminal term of the given routing policy, the route is evaluated against the next policy, if present, in the same manner as described in step 1.

Match Condition: From

The match condition `from` contains a list of match conditions to be satisfied for applying the action specified in the term. It is possible that the term doesn’t have any match condition. This indicates that all routes match this term and action is applied according to the action specified in the term.

The following table describes the match conditions supported by Contrail Networking.

Match Condition	User Input	Description
Prefix	List of prefixes to match	Each prefix in the list is represented as prefix and match type, where the prefix match type can be: • `exact` • `orlonger` • `longer` Example: 1.1.0.0/16 `orlonger` A route matches this condition if its prefix matches any of the prefixes in the list.
Community	Community string to match	Represented as either a well-known community string with `no export` or `no reoriginate`, or a string representation of a community (64512:11).
Protocol	Array of path source or path protocol to match	BGP | XMPP | StaticRoute | ServiceChain | Aggregate. A path is considered as matching this condition if the path protocol is one of protocols in the list.

Routing Policy Action and Update Action

The policy action contains two parts, action and update action.

The following table describes `action` as supported by Contrail Networking.

Action	Terminal?	Description
Reject	Yes	Reject the route that matches this term. No more terms are evaluated after hitting this term.
Accept	Yes	Accept the route that matches this term. No more terms are evaluated after hitting this term. The route is updated using the update specified in the policy action.

(Continued)

Action	Terminal?	Description
Next Term	No	This is the default action taken upon matching the policy term. The route is updated according to the update specified in the policy action. Next terms present in the routing policy are processed on the route. If there are no more terms in the policy, the next routing policy is processed, if present.

The update action section specifies the route modification to be performed on the matching route.

The following table describes `update action` as supported by Contrail Networking.

Update Action	User Input	Description
Community	List of community	As part of the policy update, the following actions can be taken for community: • Add a list of community to the existing community. • Set a list of community. • Remove a list of community (if present) from the existing community.
MED	Update the MED of the BgpPath	Unsigned integer representing the MED
local-pref	Update the local-pref of the BgpPath	Unsigned integer representing local-pref

Applying Routing Policies to Secondary Routes

A virtual network routing policy is automatically applied to secondary routes. The ability to apply routing policies to secondary routes is especially useful as a mechanism to modify routes imported from MP-BGP, including routes that are imported from the MPLS network.

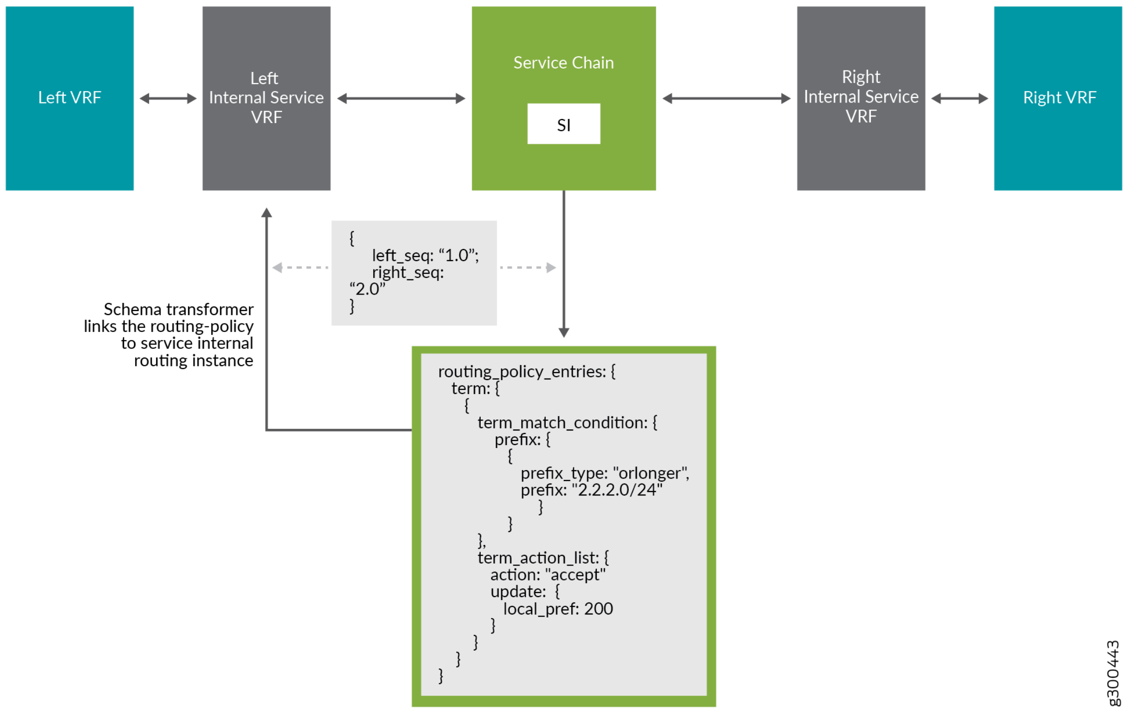
NOTE: Routing policies that are attached to service instances are applied to primary routes only. These routing policies are not applied to secondary routes.

Routing Policy Configuration

Routing policy is configured on the service instance. Multiple routing policies can be attached to a single service instance interface.

When the policy is applied on the left interface, the policy is evaluated for all the routes that are reoriginated in the left VN for routes belonging to the right VN. Similarly, the routing policy attached to the right interface influences the route reorigination in the right VN, for routes belonging to the left VN.

The following figure illustrates a routing policy configuration.


Left VRF
Left Internal Service VRF
Service Chain
SI
Right Internal Service VRF
Right VRF
Schema transformer links the routing-policy to service internal routing instance
g300443

```
{
    left_seq: “1.0”;
    right_seq:
“2.0”
}
```

```
routing_policy_entries: {
  term: {
    {
      term_match_condition: {
         prefix: {
           {
               prefix_type: "orlonger",
               prefix: "2.2.2.0/24"
                }
           }
      },
      term_action_list: {
        action: "accept"
        update:  {
          local_pref: 200
        }
      }
    }
  }
}
```

The policy sequence number specified in the routing policy link data determines the order in which the routing policy is evaluated. The routing policy link data on the service instance also specifies whether the policy needs to be applied to the left service interface, to the right service interface, or to both interfaces.

It is possible to attach the same routing policy to both the left and right interfaces for a service instance, in a different order of policy evaluation. Consequently, the routing policy link data contains the sequence number for policy evaluation separately for the left and right interfaces.

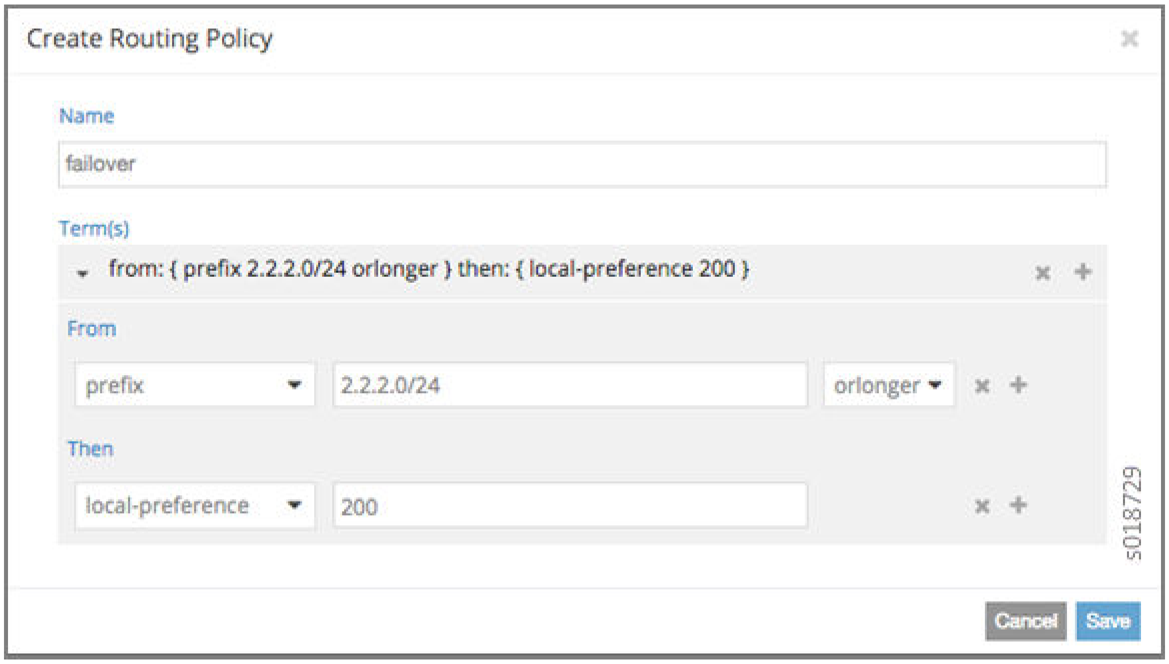
The schema transformer links the routing policy object to the internal routing instance created for the service instance. The transformer also copies the routing policy link data to ensure the same policy order.

Configuring and Troubleshooting Routing Policy

This section shows how to create a routing policy for service chains and how to validate the policy.

Create Routing Policy

First, create the routing policy, **Configure** > **Networking** > **Routing** > **Create** >**Routing Policy**. See the following example.


Create Routing Policy

Name: failover

Term(s): from: { prefix 2.2.2.0/24 orlonger } then: { local-preference 200 }

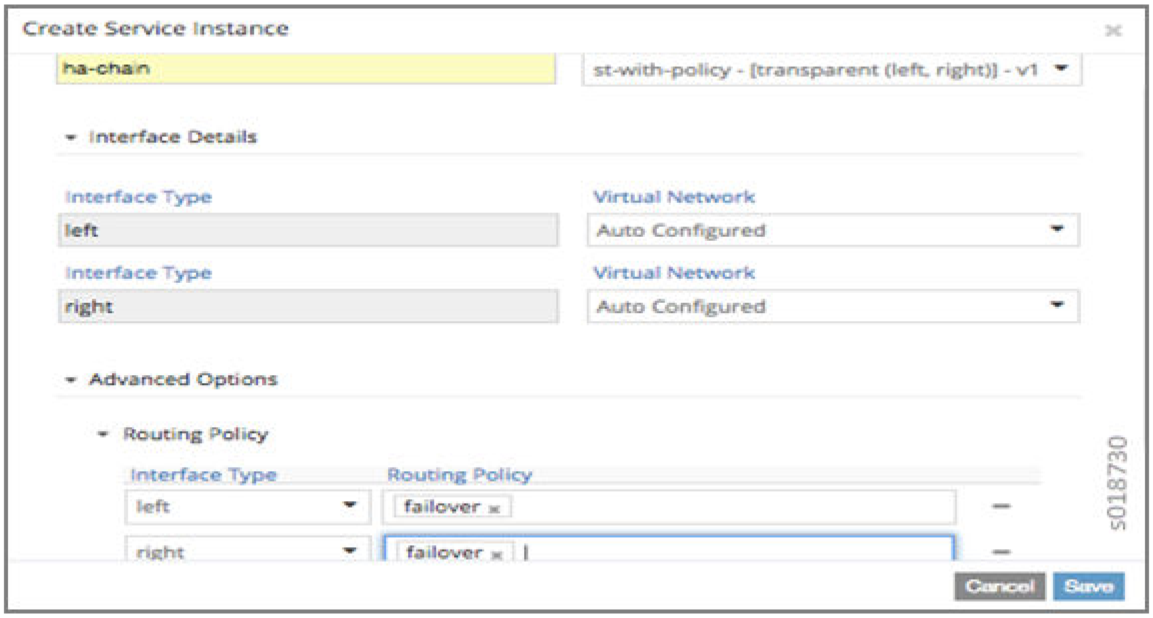
From: prefix 2.2.2.0/24 orlonger

Then: local-preference 200

NOTE: The Contrail Web UI and REST APIs enable you to configure a BGP routing policy and then assign it to a virtual network, but the routing policy will not be applied if the virtual network is attached to an L3VPN.

Configure Service Instance

Create a service instance and attach the routing policy to both the left and right interfaces. The order of the policy is calculated by the UI, based on the order of the policy specified in the list.


Create Service Instance

ha-chain | st-with-policy - [transparent (left, right)] - v1

Interface Details

Interface Type	Virtual Network
left	Auto Configured
right	Auto Configured

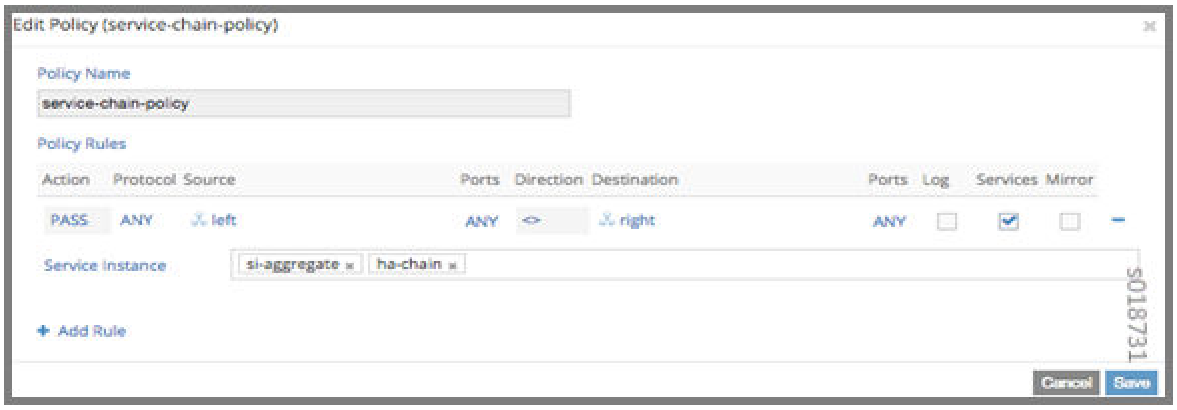
Advanced Options

Routing Policy

Interface Type	Routing Policy
left	failover
right	failover

Configure the Network Policy for the Service Chain

At **Edit Policy**, create a policy for the service chain, see the following example.


Edit Policy (service-chain-policy)

Policy Name: service-chain-policy

Policy Rules

Action	Protocol	Source	Ports	Direction	Destination	Ports	Log	Services	Mirror
PASS	ANY	left	ANY	<>	right	ANY	☐	☑	☐

Service Instance: si-aggregate, ha-chain

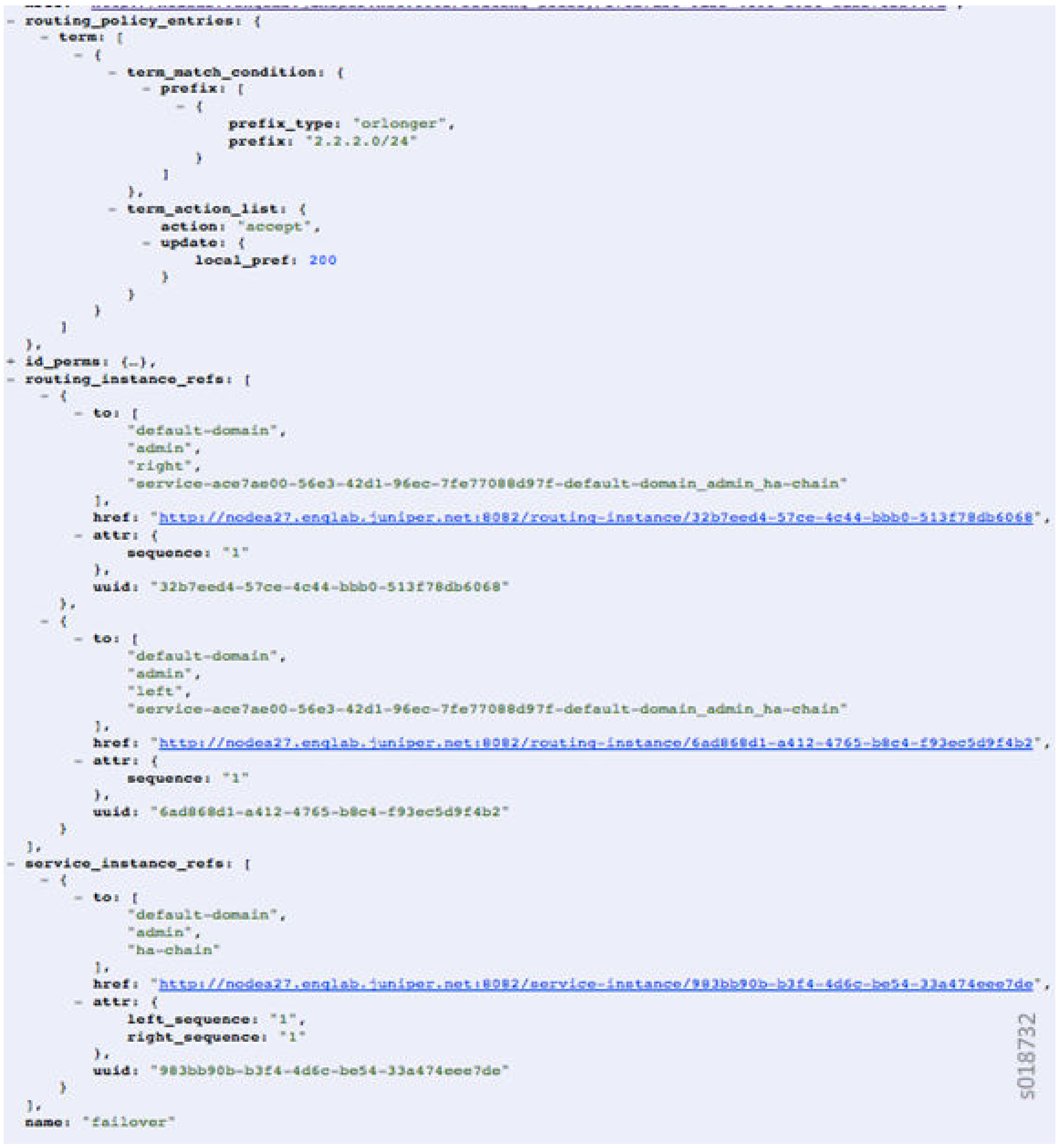
Using a VNC Script to Create Routing Policy

The following example shows use of a VNC API script to create a routing policy.

```
from vnc_api.vnc_api import *
vnc_lib = VncApi("admin", "<password>", "admin")
project=vnc_lib.project_read(fq_name=["default-domain", "admin"])
routing_policy=RoutingPolicy(name="vnc_3", parent_obj=project)
policy_term=PolicyTermType()
policy_statement=PolicyStatementType()

match_condition=TermMatchConditionType(protocol=["bgp"], community="22:33")
prefix_match=PrefixMatchType(prefix="1.1.1.0/24", prefix_type="orlonger")
match_condition.set_prefix([prefix_match])

term_action=TermActionListType(action="accept")
action_update=ActionUpdateType(local_pref=101, med=10)
add_community=ActionCommunityType()
comm_list=CommunityListType(["11:22"])
add_community.set_add(comm_list)
action_update.set_community(add_community)
term_action.set_update(action_update)

policy_term.set_term_action_list(term_action)
policy_term.set_term_match_condition(match_condition)

policy_statement.add_term(policy_term)
routing_policy.set_routing_policy_entries(policy_statement)
vnc_lib.routing_policy_create(routing_policy)
```

Verify Routing Policy in API Server

You can verify the service instance references and the routing instance references for the routing policy by looking in the API server configuration database. See the following example.



```
- routing_policy_entries: {
  - term: [
    - {
      - term_match_condition: {
        - prefix: [
          - {
                prefix_type: "orlonger",
                prefix: "2.2.2.0/24"
            }
          ]
        },
      - term_action_list: {
          action: "accept",
        - update: {
              local_pref: 200
          }
        }
      }
    ]
  },
+ id_perms: {…},
- routing_instance_refs: [
  - {
    - to: [
          "default-domain",
          "admin",
          "right",
          "service-ace7ae00-56e3-42d1-96ec-7fe77088d97f-default-domain_admin_ha-chain"
      ],
      href: "http://nodea27.englab.juniper.net:8082/routing-instance/32b7eed4-57ce-4c44-bbb0-513f78db6068",
    - attr: {
          sequence: "1"
      },
      uuid: "32b7eed4-57ce-4c44-bbb0-513f78db6068"
    },
  - {
    - to: [
          "default-domain",
          "admin",
          "left",
          "service-ace7ae00-56e3-42d1-96ec-7fe77088d97f-default-domain_admin_ha-chain"
      ],
      href: "http://nodea27.englab.juniper.net:8082/routing-instance/6ad868d1-a412-4765-b8c4-f93ec5d9f4b2",
    - attr: {
          sequence: "1"
      },
      uuid: "6ad868d1-a412-4765-b8c4-f93ec5d9f4b2"
    }
  ],
- service_instance_refs: [
  - {
    - to: [
          "default-domain",
          "admin",
          "ha-chain"
      ],
      href: "http://nodea27.englab.juniper.net:8082/service-instance/983bb90b-b3f4-4d6c-be54-33a474eee7de",
    - attr: {
          left_sequence: "1",
          right_sequence: "1"
      },
      uuid: "983bb90b-b3f4-4d6c-be54-33a474eee7de"
    }
  ],
  name: "failover"
```

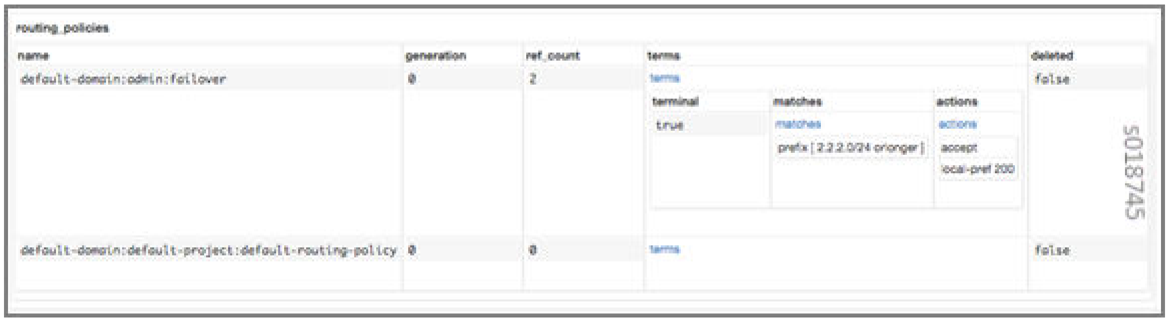
Verify Routing Policy in the Control Node

You can verify the routing policy in the control node.

Point your browser to:

`http://<control-node>:8083/Snh_ShowRoutingPolicyReq?search_string=failover`

See the following example.


routing_policies

name	generation	ref_count	terms	deleted
default-domain:admin:failover	0	2	terms terminal: true matches: prefix [2.2.2.0/24 orlonger] actions: accept, local-pref 200	false
default-domain:default-project:default-routing-policy	0	0	terms	false

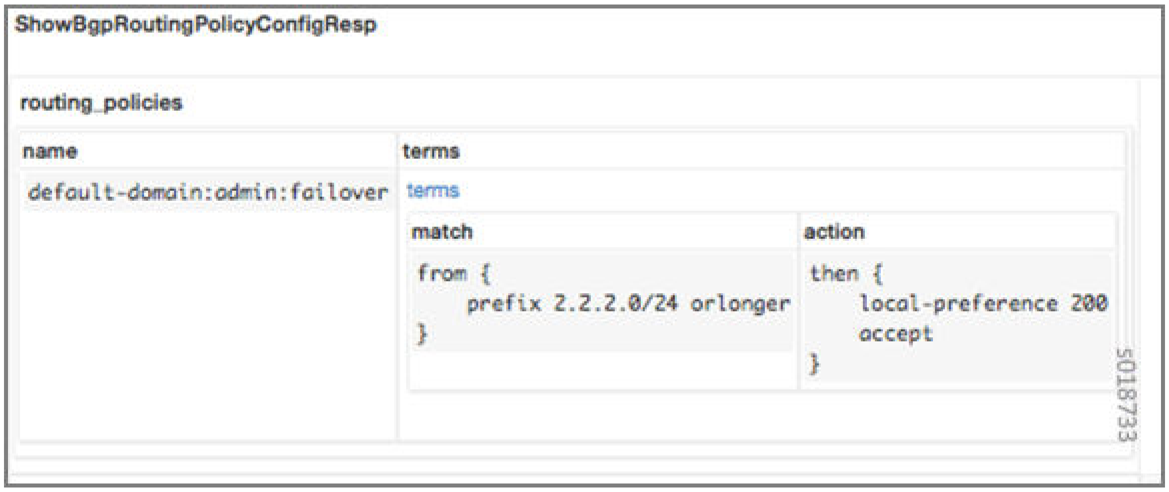
Verify Routing Policy Configuration in the Control Node

You can verify the routing policy configuration in the control node.

Point your browser to:

```
http://<control-node>:8083/Snh_ShowBgpRoutingPolicyConfigReq?search_string=failover
```

See the following example.


ShowBgpRoutingPolicyConfigResp

routing_policies

name	terms
default-domain:admin:failover	terms

match	action

from {
    prefix 2.2.2.0/24 orlonger
}
	then {
	    local-preference 200
	    accept
	}

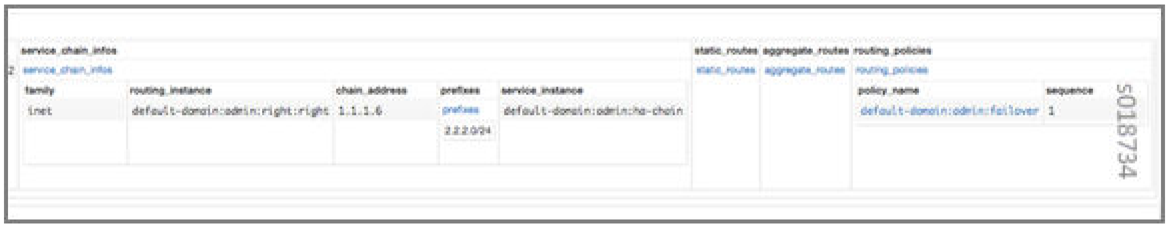
Verify Routing Policy Configuration on the Routing Instance

You can verify the routing policy configuration on the internal routing instance.

Point your browser to:

```
http://<control-node>:8083/Snh_ShowBgpInstanceConfigReq?search_string=<name-of-internal-vrf>
```

See the following example.



service_chain_infos | static_routes | aggregate_routes | routing_policies

2 service_chain_infos | static_routes | aggregate_routes | routing_policies

family	routing_instance	chain_address	prefixes	service_instance
inet	default-domain:admin:right:right	1.1.1.6	prefixes 2.2.2.0/24	default-domain:admin:ha-chain

policy_name	sequence
default-domain:admin:failover	1

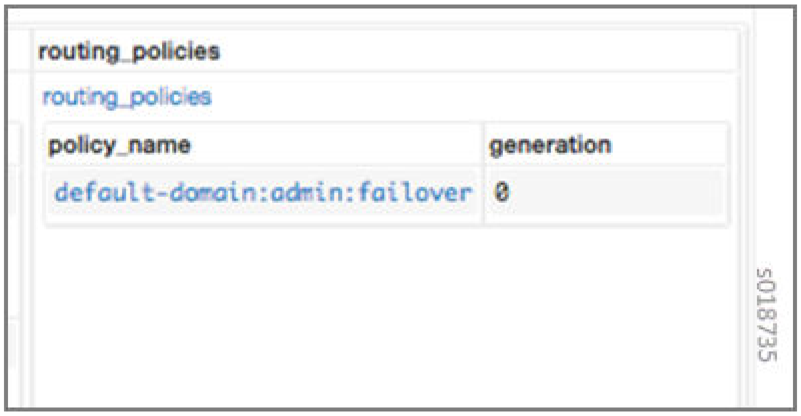
s018734

You can also verify the routing policy on the routing instance operational object.

Point your browser to:

`http://<control-node>:8083/Snh_ShowRoutingInstanceReq?x=<name-of-internal-vrf>`

See the following example.



routing_policies

routing_policies

policy_name	generation
default-domain:admin:failover	0

s018735

Control for Route Reorigination

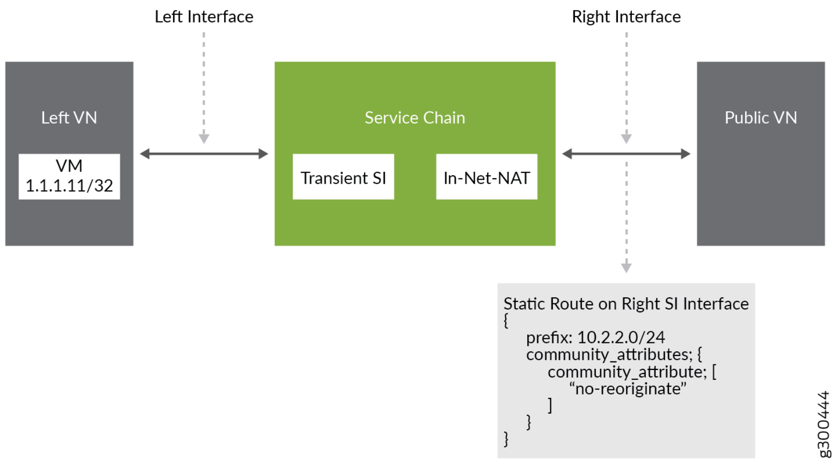
The ability to prevent reorigination of interface static routes is typically required when routes are configured on an interface that belongs to a service VM.

As an example, the following image shows a service chain that has multiple service instances, with an `in-net-nat` service instance as the last service VM, also with the right VN as the public VN.

The last service instance performs NAT by using a NAT pool. The right interface of the service VM must be configured with an interface static route for the NAT pool so that the destination in the right VN knows how to reach addresses in the NAT pool. However, the NAT pool prefix should not be reoriginated into the left VN.

To prevent route reorigination, the interface static route is tagged with a well-known BGP community called `no-reoriginate`.

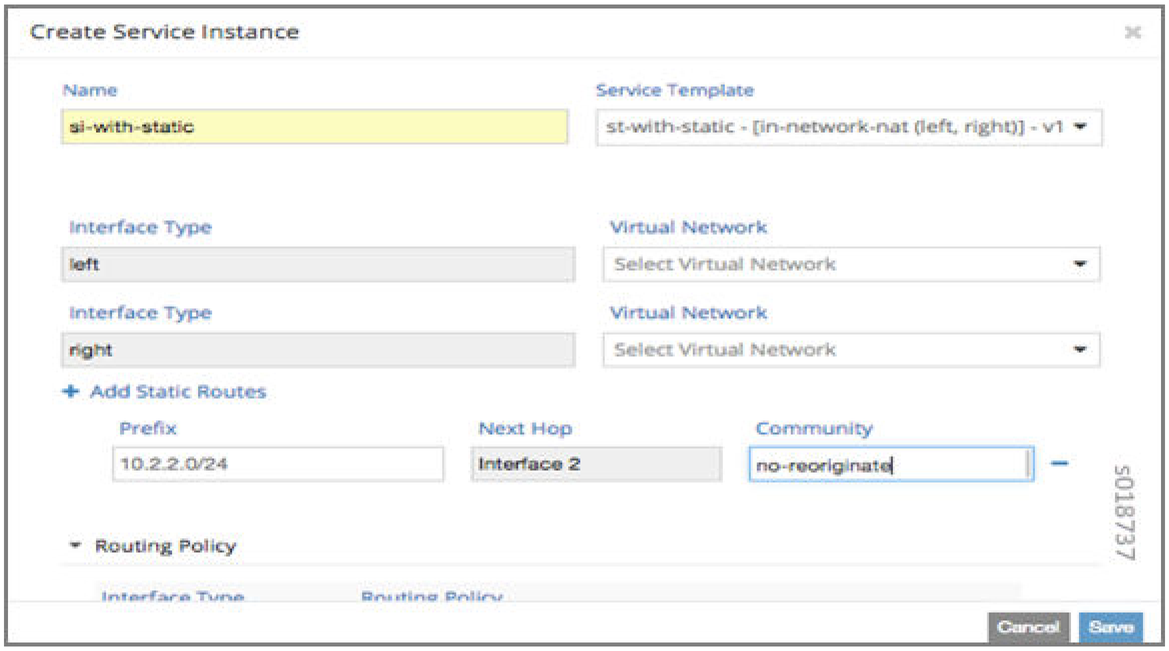
When the control node is reoriginating the route, it skips the routes that are tagged with the BGP community.



```
Static Route on Right SI Interface
{
    prefix: 10.2.2.0/24
    community_attributes; {
        community_attribute; [
            "no-reoriginate"
        ]
    }
}
```

Configuring and Troubleshooting Reorigination Control

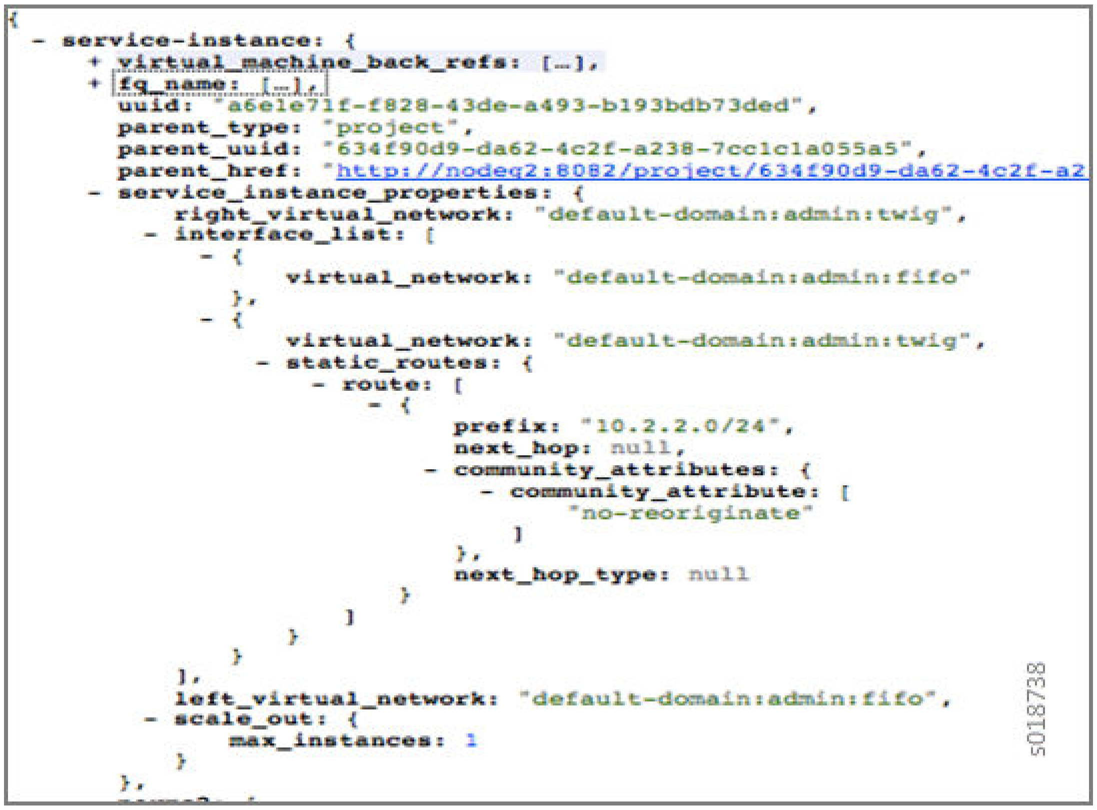
The community attribute on the static routes for the interface static route of the service instance is specified during creation of the service instance. See the following example.



Name	si-with-static
Service Template	st-with-static - [in-network-nat (left, right)] - v1
Interface Type	left
Virtual Network	Select Virtual Network
Interface Type	right
Virtual Network	Select Virtual Network

Prefix	Next Hop	Community
10.2.2.0/24	Interface 2	no-reoriginate

Use the following example to verify that the service instance configuration object in the API server has the correct community set for the static route. See the following example.



```
{
  - service-instance: {
      + virtual_machine_back_refs: […],
      + fq_name: […],
        uuid: "a6e1e71f-f828-43de-a493-b193bdb73ded",
        parent_type: "project",
        parent_uuid: "634f90d9-da62-4c2f-a238-7cc1c1a055a5",
        parent_href: "http://nodeg2:8082/project/634f90d9-da62-4c2f-a2
      - service_instance_properties: {
            right_virtual_network: "default-domain:admin:twig",
          - interface_list: [
              - {
                    virtual_network: "default-domain:admin:fifo"
                },
              - {
                    virtual_network: "default-domain:admin:twig",
                  - static_routes: {
                      - route: [
                          - {
                                prefix: "10.2.2.0/24",
                                next_hop: null,
                              - community_attributes: {
                                  - community_attribute: [
                                        "no-reoriginate"
                                    ]
                                },
                                next_hop_type: null
                            }
                        ]
                    }
                }
            ],
            left_virtual_network: "default-domain:admin:fifo",
          - scale_out: {
                max_instances: 1
            }
        },
```

s018738

Filtering MP-BGP Traffic using a Routing Policy

Starting in Contrail Networking Release 21.4.L1, you can use Contrail Networking to directly apply import routing policies to MP-BGP routes only.

Routing Policies are configured in Contrail Command in the **Overlay** > **Routing** > **Create Routing Policy** page.

The behavior of route filtering for BGP-learned routes changed in Contrail Networking Release 21.4.L1:

- In Contrail Networking Release 21.4.L1 and later releases, setting the **Protocol** field in the **From** box to **bgp** configures your import routing policy to apply route manipulation to MP-BGP routes only. BGPaaS-learned routes are not identified by the routing policy.

  You can, notably, also configure a routing policy that applies to BGPaaS-learned routes only by setting the **From** field as **BGPaaS**.

  You can still create a routing policy that applies to both MP-BGP and BGPaaS-learned routes and by setting the **From** field to match on both **bgp** and **BGPaaS**.

- In Contrail Networking Release 21.4 and earlier releases, setting the **Protocol** field in the **From** box to **bgp** configures your import routing policy to apply route manipulation to MP-BGP routes and BGPaaS-learned routes.

  You can, notably, also configure a routing policy that applies to BGPaaS-learned routes only by setting the **From** field as **BGPaaS**. You do not have the option to configure a routing policy that applies to MP-BGP routes only.

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
1910	Starting with Contrail Networking Release 1910, virtual network routing policies are automatically applied to secondary routes. See "Applying Routing Policies to Secondary Routes" on page 63.
21.4.L1	Starting in Contrail Networking Release 21.4.L1, you can use Contrail Networking to directly apply import routing policies to MP-BGP routes only.

RELATED DOCUMENTATION

Creating a Routing Policy With Extended Communities in Contrail Command | 73

Creating a Routing Policy With Extended Communities in Contrail Command

Contrail Networking supports extended communities on the import routing policy function. Contrail Networking also enables you to import routing policy terms to match on extended communities and perform import routing policy actions to add, set, and remove extended communities. Filtering routes based on extended communities prevent advertising unnecessary service interface and static routes from the control node.

The following extended communities are supported:

- Route Target
- Encapsulation
- Security Group

- Origin VN
- MAC Mobility
- Load Balance
- Tag

For information on these extended communities, see BGP Extended Communities.

Creating a Routing Policy

This section shows how to create a routing policy for a virtual network with extended communities.

1. Click the **Create** button in **Overlay** > **Routing** > **Routing Policies**.
2. Enter routing policy information according to the guidelines provided in Table 4 on page 74
3. Click **Create** to create the routing policy.

   The **Routing Policies** tab is displayed listing the newly created policy.
4. Navigate to the **Overlay** > **Virtual Networks** page.
5. Select the check box for the virtual network that you want to attach the routing policy to, and click the **Edit** icon.

   The **Edit Virtual Network** page appears.
6. Scroll down to the **Routing, Bridging, and Policies** section in the **Network** tab, and select the newly created routing policy in the **Routing Policies** field.
7. Click **Save** to add the routing policy to the virtual network.

Table 4: Create Routing Policy

Field	Guidelines
Name	Enter a name for the routing policy.
Type	Select **Physical Device** or **vRouter**. You can create a routing policy for the type of device you select. Select **vRouter** to create a routing policy for a virtual network with extended communities.
Term(s)	
Community	Select the community string to match for the routing policy. The community string is represented with **accept-own**, **no-advertise**, **no-export**, **no-export-subconfed**, **no-reoriginate**..

Table 4: Create Routing Policy ***(Continued)***

Field	Guidelines
Match All	Select the check box to match all the community strings.
Extended Community	Select the extended community string to match for the routing policy.
Match All	Select the check box to match the extended community strings.
Protocol	Select the protocol for the routing policy which is an array of path source or path protocol to match. The protocols are **interface, aggregate, bgp, BGPaaS, interface-static, service-chain, service-interface, static,** and **xmpp**. A path is considered as matching this condition if the path protocol is one of protocols in the list.
Prefixes	Select a list of prefixes to match. Each prefix in the list is represented as prefix and match type, where the prefix match type can be: • `exact` • `orlonger` • `longer` Example: 10.x.x.0/16 `orlonger` A route matches this condition if its prefix matches any of the prefixes in the list.
Then	

Table 4: Create Routing Policy *(Continued)*

Field	Guidelines
Actions	Select the actions to be performed on the matching routes. The supported actions and the values are:

Action	Value
action	Reject-Reject the route that matches this term. No more terms are evaluated after hitting this term. Accept-Accept the route that matches this term. No more terms are evaluated after hitting this term. Next-This is the default action taken upon matching the policy term. The route is updated according to the update specified in the policy action. Next terms present in the routing policy are processed on the route. If there are no more terms in the policy, the next routing policy is processed, if present.
add community Add a list of community to the existing community.	The community is of type `unsigned 32 bit integer:unsigned 32 bit integer`. For example, 64512:55555.
add extended community Add a list of extended community to the existing community.	An eight octet string representation of value `type:administrator:assigned-number` where type is two octets, administrator is four octets, and assigned-number is two octets or it can be a hexadecimal representation of the community like: `0xFFffA101`.
as-path Select different AS paths to control routing decisions	Unsigned 32-bit integer representing the as-path. For example, 444.
local-preference	Unsigned 32-bit integer representing local-preference. For example, 444.

Table 4: Create Routing Policy *(Continued)*

Field	Guidelines	
	Select the local preference to distinguish routes and take further action.	
	med Select the MED of the BgpPath.	Unsigned 32-bit integer representing the MED. For example, 444.
	remove community Remove a list of community (if present) from the existing community.	The community is of type `unsigned 32 bit integer:unsigned 32 bit integer`.
	remove extended community Remove a list of extended community (if present) from the existing community.	An eight octet string representation of value `type:administrator:assigned-number` where type is two octets, administrator is four octets, and assigned-number is two octets or it can be a hexadecimal representation of the community like: `0xFFffA101`.
	set community Set a list of community.	The community is of type `unsigned 32 bit integer:unsigned 32 bit integer`.
	set extended community Set a list of extended community.	An eight octet string representation of value `type:administrator:assigned-number` where type is two octets, administrator is four octets, and assigned-number is two octets or it can be a hexadecimal representation of the community like: `0xFFffA101`.

RELATED DOCUMENTATION

Routing Policy | 59

Creating Routing Policies for QFX Series Devices in Contrail Networking

Starting with Contrail Networking Release 2005, you can create routing policies with routing policy terms supported by QFX Series devices. In releases prior to release 2005, you could not configure routing policies on physical devices. You could configure routing policies only on vRouters.

In release 2005, you can use routing policies to leak routes between logical routers set up on QFX Series devices. You must configure the routing policies with matching conditions that are supported by a QFX Series device. To leak routes across logical routers deployed in a fabric, you must configure a source logical router and one or more destination logical routers. As specified in the routing policy terms, if an incoming route meets the matching conditions, corresponding actions are performed on the routes. The accepted routes are leaked from a source logical router to all the destination logical routers.

Perform the following steps in Contrail Command to create or edit routing policies and assign routing policy terms that are supported by QFX Series devices.

1. Navigate to **Overlay**>**Routing**>**Routing Policies**.
2. Click **Create** to create a new routing policy.

   Alternatively, you can also edit an existing routing policy for a QFX Series device. To edit an existing policy, select a policy from the displayed list and click the **Edit (pencil)** icon.

   The **Create Routing Policy** page is displayed.
3. Enter routing policy information according to the guidelines provided in Table 5 on page 79.
4. Click **Create** to create the routing policy for a QFX Series device.

   The **Routing Policies** tab is displayed listing the newly created policy along with existing policies.

The routing policies are listed according to their name, type, and the routing policy terms assigned to them. The **Name** column indicates the name of the routing policy and the **Terms** column indicates the routing policy terms assigned to the routing policy. Starting with release 2005, the **Type** column indicates if a routing policy is created for a QFX Series device or a vRouter.

Table 5: Create Routing Policy for a QFX Series Device

Field	Guidelines
Name	Enter a name for the routing policy in the **Name** field.
Type	Select **Physical Device** or **vRouter**. You can create a routing policy for the type of device you select. Select **Physical Device** to create a routing policy for a QFX Series device.
Term(s)	
From	Select the matching conditions to be satisfied by the incoming routes.
Add Route filter > **Route Filter**	Enter an IP prefix address as a route filter in the **Route Filter** field.
Type	Select one or more type of prefix. If an incoming route satisfies the prefix match condition, the route is processed.
Community	Select the community string to match for the routing policy. The community string is represented with **no-advertise**, **no-export**, and **no-export-subconfed**.
Match All	Select the check box to match all the community strings.
Click **Add match condition**. The **Additional Match conditions** field is displayed. To add more match conditions, click on **Add match condition**.	Select a match condition from the list. A route must match the criteria of the match condition you selected. You can define one or more match conditions. If a route matches all match conditions, one or more actions are applied to the route.

Table 5: Create Routing Policy for a QFX Series Device ***(Continued)***

Additional Match conditions

Table 5: Create Routing Policy for a QFX Series Device *(Continued)*

AS-path	Select **AS-path** from the list. In the **Type/Value** field, enter the name of the path regular expression of an autonomous systems (AS). In the BGP routes, the AS path that matches the regular expression are processed.
External	Select **External** from the list. In the **Type/Value** field, select the type of OSPF route: OSPF Type1 or OSPF Type2. External OSPF routes, including routes exported from one level to another are processed.
Family	Select **Family** from the list. In the **Type/Value** field, select the type of BGP family from the list to which the route belongs: • `evpn` • `inet` • `inet-vpn`
local-pref	Select **local-pref** from the list. In the **Type/Value** field, enter a preference value between 0-4,294,967,295 (2^32 – 1). A route that matches the BGP local preference attribute, is processed.
NLRI	Select **NLRI** from the list. In the **Type/Value** field, select a NLRI route type from the list: **Type 1** to **Type 10**. You can specify multiple route types in a single policy.

Prefix List	Select **Prefix List** from the list. In the **Type/Value** field, select a prefix from the list of prefixes in the prefix list. You must select a prefix length qualifier from the list beside the **Type/Value** field: • `exact` • `orlonger` • `longer` Example: 10.1.0.0/16 `orlonger` A route matches this condition if its prefix matches any of the prefixes in the list.
Protocol	Select **Protocol** from the list. In the **Type/Value** field, select the name of the protocol from which the route was learned or to which the route is being advertised. The protocols are **aggregate**, **bgp**, **direct**, **evpn**, **ospf**, **ospf3**, **pim**, and **static**.

Table 5: Create Routing Policy for a QFX Series Device ***(Continued)***

Then

Select the actions to be performed on the matching routes. The supported actions and the values are:

Action	Value
action	Reject-Reject the route that matches this term. No more terms are evaluated after hitting this term. Accept-Accept the route that matches this term. No more terms are evaluated after hitting this term. Next-This is the default action taken upon matching the policy term. The route is updated according to the update specified in the policy action. Next terms present in the routing policy are processed on the route. If there are no more terms in the policy, the next routing policy is processed, if present.
add community Add a list of community to the existing community.	The community is of type `unsigned 32 bit integer:unsigned 32 bit integer`. For example, 64512:55555.
as-path-extend Extract the last AS number in the existing AS path and affix that AS number to the beginning of the AS path n times, where n is a number from 1 through 32	Unsigned 32-bit integer representing the as-path-extend . For example, 444.
as-path-prepend Affix one or more AS numbers at the beginning of the AS path. The AS numbers are added after the local AS number has been added to the path.	Unsigned 32-bit integer representing the as-path-prepend . For example, 444.
external	Unsigned 32-bit integer representing the as-path .

Set the acceptable external routes exported by OSPF. You must specify the OSPF type.	For example, Type 1 or Type 2.
local-preference Select the local preference to distinguish routes and take further action.	Unsigned 32-bit integer representing local-preference. For example, 444.
med Select the MED of the BgpPath.	Unsigned 32-bit integer representing the MED. For example, 444.
remove community Remove a list of community (if present) from the existing community.	The community is of type `unsigned 32 bit integer:unsigned 32 bit integer`.
set community Set a list of community.	The community is of type `unsigned 32 bit integer:unsigned 32 bit integer`.

Change History Table

Feature support is determined by the platform and release you are using. Use Feature Explorer to determine if a feature is supported on your platform.

Release	Description
2005	Starting with Contrail Networking Release 2005, you can create routing policies with routing policy terms supported by QFX Series devices

RELATED DOCUMENTATION

Logical Router Interconnect

Routing Policy | 59

Routing Policy Match Conditions

Actions in Routing Policy Terms

Service Instance Health Checks

IN THIS SECTION

Contrail Networking enables you to use a service instance health check to determine the liveliness of a service provided by a virtual machine (VM).

Health Check Object

IN THIS SECTION

Health Check Overview

The service instance health check is used to determine the liveliness of a service provided by a VM, checking whether the service is operationally up or down. The vRouter agent uses ping and an HTTP URL to the link-local address to check the liveliness of the interface.

If the health check determines that a service is no longer operational, it removes the routes for the VM, thereby disabling packet forwarding to the VM.

The service instance health check is used with service template version 2.

Health Check Object Configuration

Table 6 on page 85 shows the configurable properties of the health check object.

Table 6: Health Check Configurable Parameters

Field	Description
`- enabled`	Indicates that health check is enabled. The default is `False`.
`- health-check-type`	Indicates the health check type: `link-local`, `end-to-end`, `bgp-as-a-service`, and so on.. The default is `link-local`.
`- monitor-type`	The protocol type to be used: `PING` or `HTTP`.
`- delay`	The delay, in seconds, to repeat the health check.
`- timeout`	The number of seconds to wait for a response.
`- max-retries`	The number of retries to attempt before declaring an instance health down.
`- http-method`	When the monitor protocol is HTTP, the type of HTTP method used, such as GET, PUT, POST, and so on.
`- url-path`	When the monitor protocol is HTTP, the URL to be used. For all other cases, such as ICMP, the destination IP address.
`- expected-codes`	When the monitor protocol is HTTP, the expected return code for HTTP operations.

Health Check Modes

The following modes are supported for the service instance health check:

- `link-local`—A local check for the service VM on the vRouter where the VM is running. In this case, the source IP of the packet is the service chain IP.
- `end-to-end`—A remote address or URL is provided for a service health check through a chain of services. The destination of the health check probe is allowed to be outside the service instance. However, the health check probe must be reachable through the interface of the service instance where the health check is attached. The end-to-end health check probe is transmitted all the way to the actual destination outside the service instance. The response to the health check probe is received and processed by the service health check to evaluate the status.

  Restrictions include:

  - This check is applicable for a chain where the services are not scaled out.
  - When this mode is configured, a new health check IP is allocated and used as the source IP of the packet.
  - The health check IP is allocated per `virtual-machine-interface` of the service VM where the health check is attached.
  - The agent relies on the `service-health-check-ip` flag to use as the source IP.

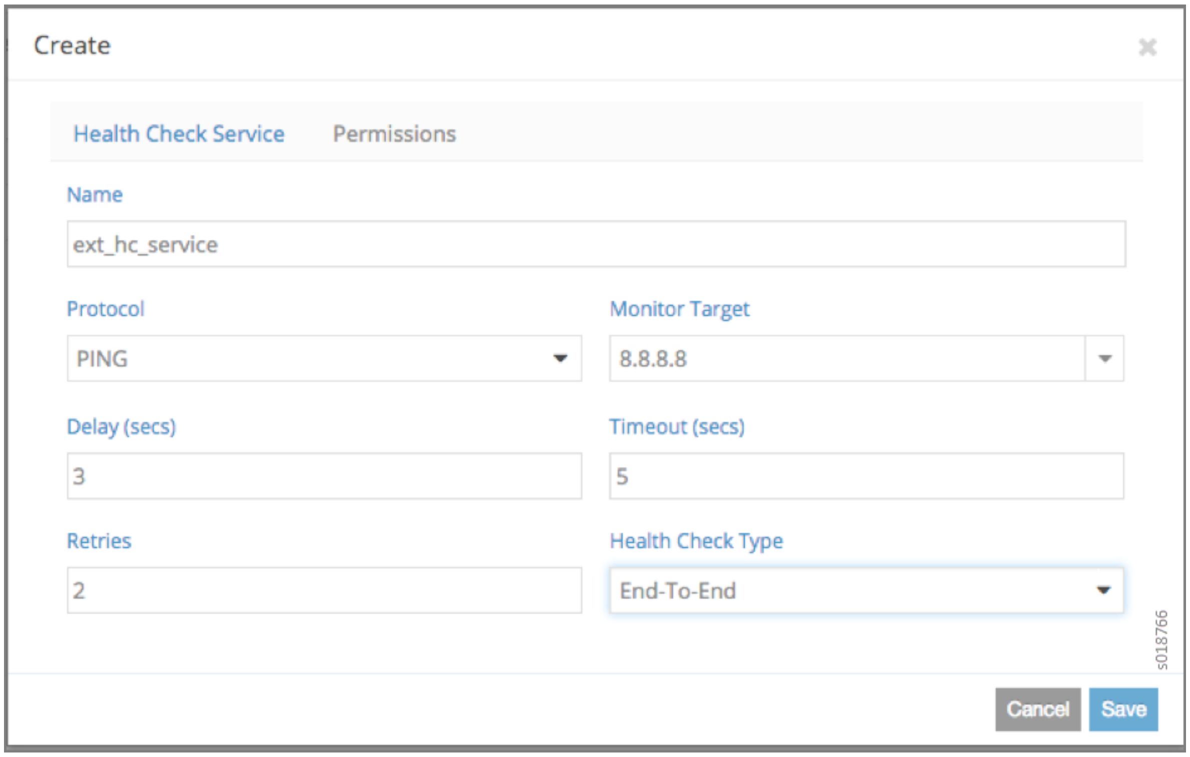
  NOTE: Contrail Networking supports a segment-based health check for transparent service chain.

Creating a Health Check with the Contrail Web UI

To create a health check with the Contrail Web UI:

1. Navigate to **Configure** > **Services** > **Health Check Service**, and click to open the **Create** screen. See Figure 8 on page 87.

Figure 8: Create Health Check Screen



Name: ext_hc_service
Protocol: PING
Monitor Target: 8.8.8.8
Delay (secs): 3
Timeout (secs): 5
Retries: 2
Health Check Type: End-To-End

2. Complete the fields to define the permissions for the health check, see Table 7 on page 87.

Table 7: Create Health Check Fields

Field	Description
Name	Enter a name for the health check service you are creating.
Protocol	Select from the list the protocol to use for the health check, PING, HTTP, BFD, and so on.
Monitor Target	Select from the list the address of the target to be monitored by the health check.
Delay (secs)	The delay, in seconds, to repeat the health check.
Timeout (secs)	The number of seconds to wait for a response.

Table 7: Create Health Check Fields *(Continued)*

Field	Description
Retries	The number of retries to attempt before declaring an instance health down.
Health Check Type	Select from the list the type of health check—link-local, end-to-end, segment-based, bgp-as-a-service, and so on.

Using the Health Check

A REST API can be used to create a health check object and define its associated properties, then a link is added to the VM interface.

The health check object can be linked to multiple VM interfaces. Additionally, a VM interface can be associated with multiple health check objects. The following is an example:

```
HealthCheckObject 1 ---------------- VirtualMachineInterface 1 ----------------
HealthCheckObject 2
      |
      |
VirtualMachineInterface 2
```

Health Check Process

The Contrail vRouter agent is responsible for providing the health check service. The agent spawns a Python script to monitor the status of a service hosted on a VM on the same compute node, and the script updates the status to the vRouter agent.

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

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

NOTE: Running health check creates flow entries to perform translation from underlay to overlay. Consequently, in a heavily loaded environment with a full flow table, it is possible to observe false failures.

Bidirectional Forwarding and Detection Health Check over Virtual Machine Interfaces

Contrail Networking supports BFD-based health checks for VMIs.

Health check for VMIs is already supported as poll-based checks with ping and curl commands. When enabled, these health checks run periodically, once every few seconds. Consequently, failure detection times can be quite large, always in seconds.

Health checks based on the BFD protocol provide failure detection and recovery in sub-second intervals, because applications are notified immediately upon BFD session state changes.

If BFD-based health check is configured, whenever a BFD session status is detected as `Up` or `Down` by the health-checker, corresponding logs are generated.

Logging is enabled in the contrail-vrouter-agent.conf file with the log severity level `SYS_NOTICE`.

You can view the log file in the location **/var/log/contrail/contrail-vrouter-agent.log**

Snippet of sample log message related to BFD session events

```
2019-02-26 Tue 14:38:49:417.479 SYS_NOTICE BFD session Down  interface: test-bfd-hc-vmi.st2
vrf: default-domain:admin:VN.hc.st2:VN.hc.st2
2019-02-26 Tue 14:38:49:479.733 PST SYS_NOTICE BFD session Up  interface: test-bfd-hc-vmi.st2
vrf: default-domain:admin:VN.hc.st2:VN.hc.st2
```

Bidirectional Forwarding and Detection Health Check for BGPaaS

Contrail Networking supports BFD-based health check for BGP as a Service (BGPaaS) sessions.

This health check should not be confused with the BFD-based health check over VMIs feature. The BFD-based health check for VMIs cannot be used for a BGPaaS session, because the session shares a tenant destination address over a set of VMIs, with only one virtual machine interface active at any given time.

When the BFD-based health check for BGP as a Service (BGPaaS) is configured, any time a BFD-for-BGP session is detected as down by the health-checker, corresponding logs and alarms are generated.

When the BFD-based health check is configured, whenever a BFD session is detected as `Up` or `Down` by the health-checker, corresponding logs are generated.

Logging for this scenario is enabled in the contrail-vrouter-agent.conf file with the log level `SYS_NOTICE`.

To enable this health check, configure the `ServiceHealthCheckType` property and associate it with a bgp-as-a-service configuration object. This can also be accomplished in the Contrail Web UI.

Health Check of Transparent Service Chain

Contrail Networking enhances service chain redundancy by implementing an end-to-end health check for the transparent service chain. The service health check monitors the status of the service chain and if there is a failure, the control node no longer considers the service chain as a valid next hop, triggering traffic failover.

A segment-based health check is used to verify the health of a single instance in a transparent service chain. The user creates a service-health-check object, with type segment-based, and attaches it to either the left or right interface of the service instance. The service health check packet is injected to the interface to which it is attached. When the packet comes out of the other interface, a reply packet is injected on that interface. If health check requests fail after 30-second retries, the service instance is considered unhealthy and the service VLAN routes of the left and right interfaces are removed. When the agent receives health check replies successfully, it adds the retracted routes back onto both interfaces, which triggers the control node to start reoriginating routes to other service instances on that service chain.

For more information, see https://github.com/tungstenfabric/tf-specs/blob/master/transparent_sc_health_check.md

Service Instance Fate Sharing

Contrail Networking supports service instance (SI) fate sharing that prevents a SI failure from causing a null route. In earlier releases, when an SI fails, the service chain continues to forward packets and routes reoriginate on both sides of the service chain. The packets are dropped in the SI or by the vRouter causing a null route.

As part of SI fate sharing in Contrail Networking, a gateway node automatically reroutes traffic to an alternate cluster and stops the service chain. If one or more than one SI in a service chain fails, routes on both sides of the service chain stops reoriginating and routes automatically converge to a backup service chain that is part of another Contrail cluster.

Contrail Networking uses **segment-based** health check to verify the health of a SI in a service chain. To identify a failure of an SI, segment-based health check is configured either on the egress or ingress interface of the SI. When SI health check fails, the vRouter agent drops an SI route or a connected route. A connected route is also dropped if the vRouter agent restarts due to a software failure, when a compute node reboots, or when long-lived graceful restart (LLGR) is not enabled. You can detect an SI failure by keeping track of corresponding connected routes of the service chain address.

NOTE: When an SI is scaled out, the connected route for an SI interface goes down only when all associated VMs have failed.

The control node uses the `service-chain-id` in `ServiceChainInfo` to link all SIs in a service chain. When the control node detects that any SI of the same service-chain-id is down, it stops reoriginating routes in egress and ingress directions for all SIs. The control node reoriginates routes only when the connected routes of all the SIs are up.

ECMP Support in Service Chain

IN THIS SECTION

Equal-cost multipath (ECMP) can be used to distribute traffic across VMs.

Service Chain with Equal-Cost Multipath in Active-Active Mode

To support ECMP in the service chain, create multiple port tuples within the same service instance. The labels should be the same for the VM ports in each port tuple. For example, if port tuple 1 uses the labels `left` and `right`, then port tuple 2 in the same service instance should also use the labels `left` and `right` for its ports.

When there are multiple port tuples, the default mode of operation is `active-active`.

Service Chain with Health Check

Service chain Version 2 also allows service instance health check configuration on a per interface label. This is used to monitor the health of the service.

For more information about the service instance health check, see "Service Instance Health Checks" on page 84.

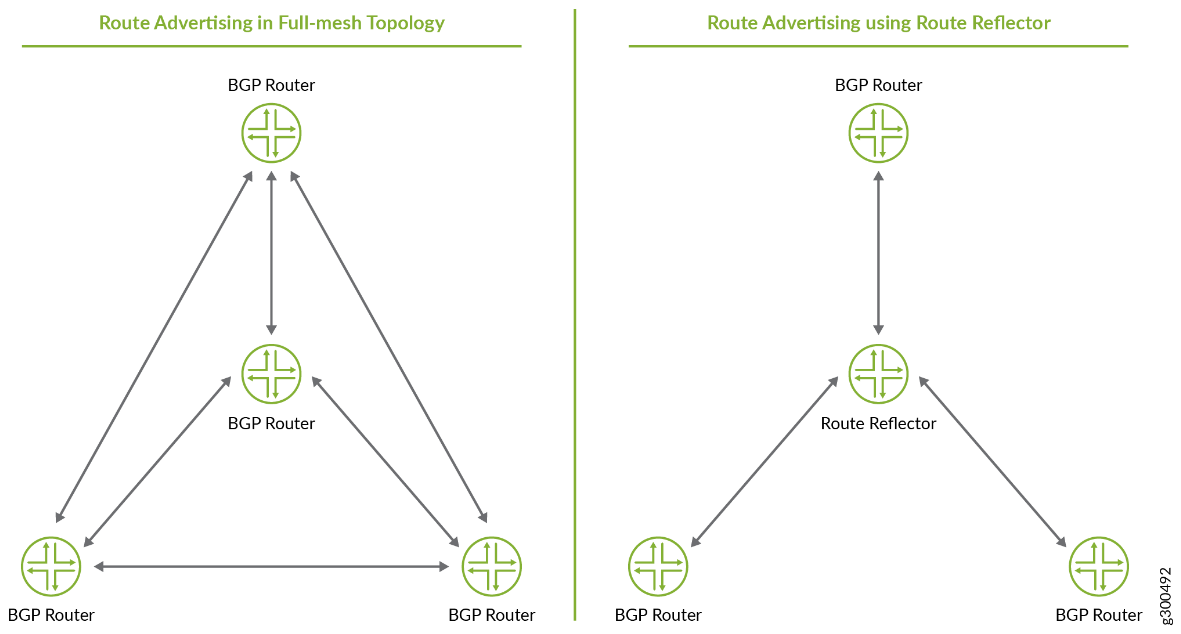
Route Reflector Support in Contrail Control Node

IN THIS SECTION

Contrail Networking supports Route Reflector (RR) functionality in the Control node for Internal Border Gateway Protocol (iBGP) peers. Route reflection is a BGP feature that enables BGP routers to acquire route information from one iBGP router and reflect or advertise the information to other iBGP peers in the same autonomous system (AS).

In a large scale AS, deploying BGP routers peering with the Control Node in a full-mesh topology affects scalability and leads to maintenance and configurational issues. The issues are caused while exchanging large volumes of routing information and maintaining connectivity among a large number of devices in the AS. A Route Reflector provides a scalable alternative to full-mesh internal BGP peering. See Figure 9 on page 93.

Figure 9: Advantage of BGP Route Reflector over Full-mesh Topology



You can create any number of RRs in the network. Each RR must have a unique cluster ID to prevent looping among the RRs.

Contrail nodes support Edge Reflection Multicasting Virtual Private Networking (ERMVPN) of the BGP Address Family. As non-Contrail nodes do not support ERMVPN, RR is configured separately for Contrail nodes, and external BGP speakers. If a single RR is deployed between Contrail nodes and non-Contrail nodes, the RR cannot advertise ERMVPN routes to the Contrail nodes. Contrail deploys a separate RR peering session among Contrail nodes that supports ERMVPN and helps in propagating routes among all Contrail nodes.

Benefits of RRs in Contrail

- RRs can be deployed at multiple locations in the network, which helps to scale the BGP network at lower cost.
- The RR feature conserves data center rack space by replacing physical route reflectors.

RELATED DOCUMENTATION

Configuring Route Reflectors from Contrail Command | 94

Configuring Route Reflectors from Contrail Command

Contrail Networking enables you to configure a control node as a route reflector from the Contrail Command user interface (UI).

Follow these steps to configure a route reflector from Contrail Command UI:

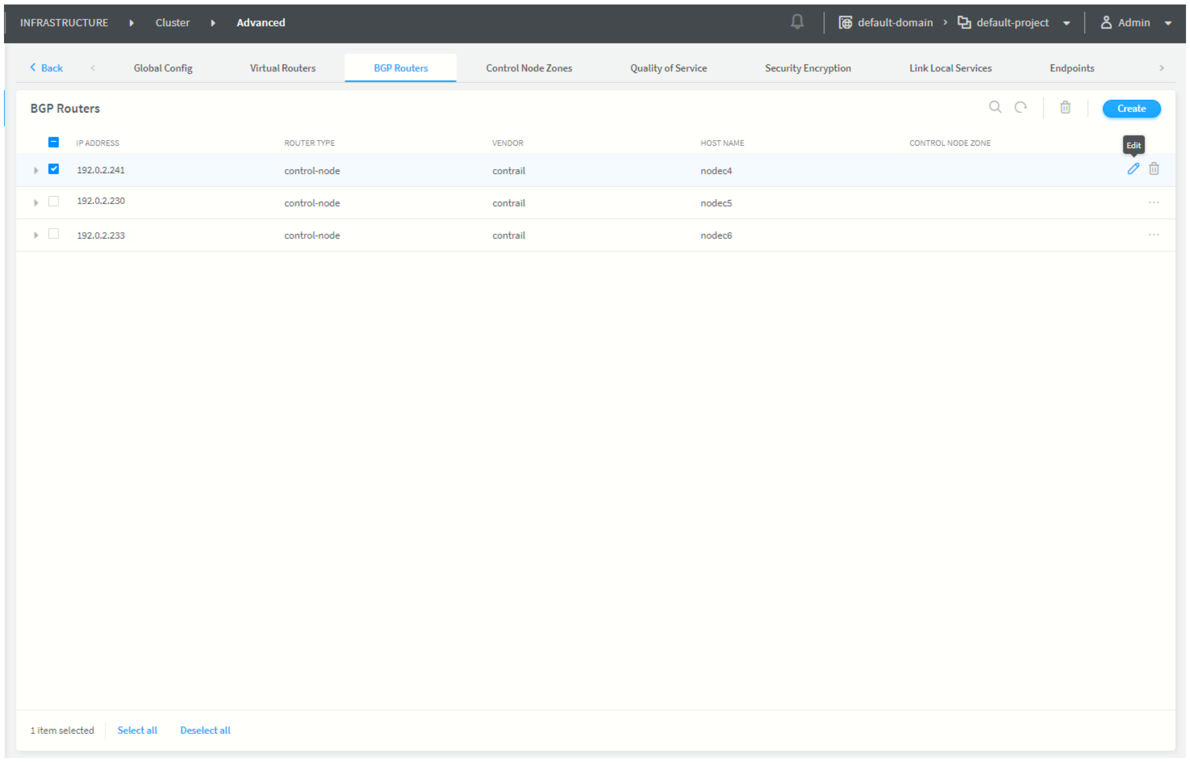
1. Click the **Infrastructure > Cluster** page.

   The **Overview** tab is displayed.
2. Click the **Advanced Options** tab.

   The **Global Config** tab is displayed, which lists all system configuration information.
3. Click the **BGP Routers** tab.

   A list of control nodes are displayed. See Figure 10 on page 95.

Figure 10: BGP Routers List



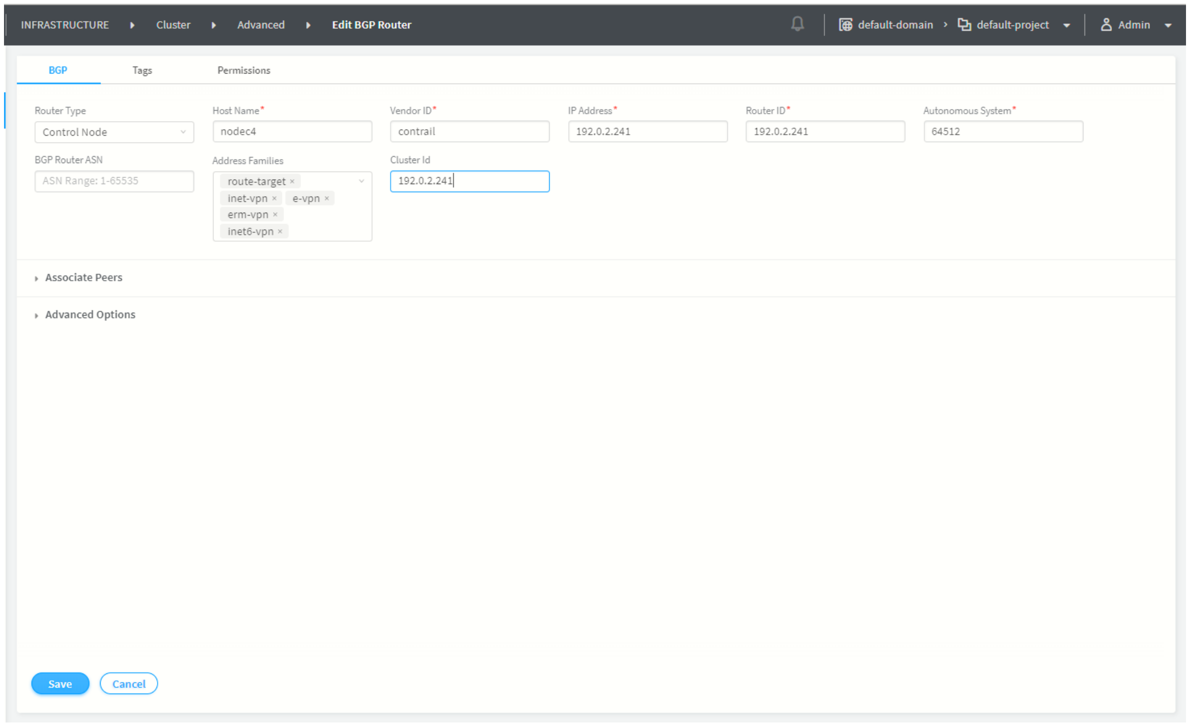
IP ADDRESS	ROUTER TYPE	VENDOR	HOST NAME	CONTROL NODE ZONE
192.0.2.241	control-node	contrail	nodec4	
192.0.2.230	control-node	contrail	nodec5	
192.0.2.233	control-node	contrail	nodec6	

4. Select the desired control node from the check box and click the **Edit** icon.

   The **BGP** tab in the **Edit BGP Router** page is displayed.

5. To configure the control node as route reflector, you must assign a **Cluster ID** value in IPv4 format. See Figure 11 on page 96.

Figure 11: Add Cluster ID value



Router Type	Host Name*	Vendor ID*	IP Address*	Router ID*	Autonomous System*
Control Node	nodec4	contrail	192.0.2.241	192.0.2.241	64512
BGP Router ASN	Address Families	Cluster Id			
ASN Range: 1-65535	route-target, inet-vpn, e-vpn, erm-vpn, inet6-vpn	192.0.2.241			

6. Click **Save**.

   The **Cluster ID** information is saved and the **BGP Routers** tab is displayed.

The selected control node starts functioning as a route reflector.

RELATED DOCUMENTATION

BGP as a Service

IN THIS SECTION

Understanding BGP as a Service

IN THIS SECTION

The BGP as a Service (BGPaaS) feature allows a guest virtual machine (VM) to place routes in its own virtual routing and forwarding (VRF) instance using BGP.

Contrail BGPaaS Features

Using BGPaaS with Contrail Networking requires the guest VM to have connectivity to the control node and to be able to advertise routes into the VRF instance.

With the BGPaaS feature:

- The vRouter agent is able to accept BGP connections from the VMs and proxy them to the control node.
- The vRouter agent always selects one of the control nodes that it is using as an XMPP server.

Contrail Networking provides route export functionality for BGPaaS sessions. The next hop for all routes advertised to the tenant VM is set to the default gateway address of the subnet of the tenant VM. This allows the tenant BGP implementation to be relatively simple, by not requiring support for recursive resolution of BGP next hops.

The BGPaaS object is associated with a virtual machine interface, not just a virtual machine (VM), which enables a tenant VM to have BGP sessions in multiple virtual networks, if required.

BGPaaS in Contrail Networking has the following features:

- By default, all BGPaaS sessions are configured to have bidirectional exchange of routes. The Boolean property `bgpaas-suppress-route-advertisement` ensures no advertisement of routes to the tenant VM.

- If inet6 routes are being advertised to the tenant VM, they are advertised with the IPv6 subnet's default gateway address as the BGP next hop. A Boolean property, `bgpaas-ipv4-mapped-ipv6-nexthop`, causes the IPv4 subnet's default gateway, in IPv4-mapped IPv6 format, to be used instead as the next hop.
- If multiple tenant VMs in the same virtual network have BGPaaS sessions and they use eBGP, the standard BGP AS path loop prevention rules prevent routes advertised by one tenant VM from being advertised to the other tenant VMs. The `as-override` field, added to the existing `BgpSessionAttributes` in the BGPaaS object, causes the control node to replace the AS number of the tenant VM with it's own AS number, when advertising routes learned from a tenant VM to another tenant VM in the same virtual network. The tenant VM does not need to implement any new functionality.

Contrail Networking provides support for high availability (HA) architectures, BGPaaS supports control node zone selection, with options available to configure BGPaaS control node zone peers.

This capability enables you to set up primary and secondary control node zones, which can have one or more control nodes. The reason for this is because BGPaaS is often being relied upon to provide routing to and from VNFs, which are comprised of several nodes across different computes, and the VNFs usually rely upon two BGP peers for HA. These control node zone features increase the robustness and failover capabilities for BGPaaS in Contrail.

The following are caveats:

- BGP sessions must use IPv4 transport.
- The VNF must support RFC 2545, *Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing*, to carry IPv6 routes over the IPv4 peer.
- Only IPv4 (inet) and IPv6 (inet6) address families are supported.

The following features are supported in Contrail Networking for BGPaaS configuration:

- Global-System-Config has an option to add, modify, or delete control node zones
- Control-Node-Zone has an option to add, modify, or delete control nodes
- Control node has an option to add, modify or delete a control node zone and it can have only one control node zone
- BGPaaS has an option to add, modify, or delete a primary or secondary control node zone
- If control node zone has more than one control-node, selection of control-node for BGP Peering is random in a control node zone
- Using just one control node in each zone, VNF can predictably establish bgp-peering to that particular control node.

BGPaaS Use Cases

This section provides example scenarios for implementing BGPaaS with Contrail.

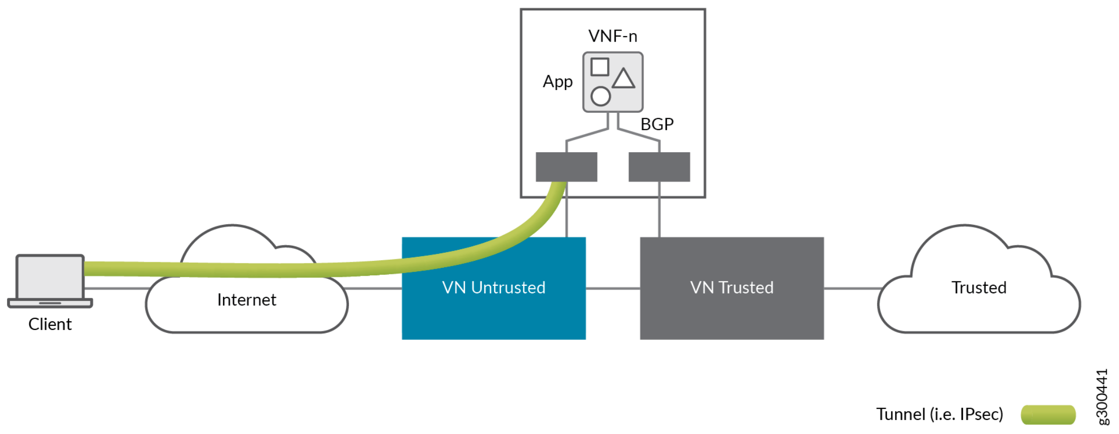
Dynamic Tunnel Insertion Within a Tenant Overlay

Various applications need to insert dynamic tunnels into virtual networks. Virtual network functions (VNFs) provide the function of tunnel termination. Tunnel termination types vary across application types, such as business VPN, mobility small site backhaul, VPC, and the like. The key requirement is that tunnels need to insert dynamically new network reachability information into the virtual network. The predominant methods of tunnel network reachability insertion use BGP.

BGPaaS allows the migration of brownfield VNFs into Contrail, preserving the application behavior and requirement for BGP, without rewriting the application.

Figure 12 on page 99 shows the need to insert a dynamic tunnel into a virtual network.

Figure 12: Dynamic Tunnel Insertion



Dynamic Network Reachability of Applications

The Domain Name System (DNS) is a widespread application that uses BGP as a mechanism to tune reachability of its services, based on metrics such as load, maintenance, availability, and the like. As DNS services are migrated to environments using overlays, a mechanism to preserve the existing application behavior and requirements is needed, including the ability to announce and withdraw reachability to the available application.

This requirement is not limited to DNS. Other applications, such as virtualized evolved packet core (vEPC) and others, use BGP as a mechanism for network reachability based on availability and load.

Liveness Detection for High Availability

Various keepalive mechanisms for tenant reachability have been provided by network components such as BGP, OSPF, PING, VRRP, BFD, or application-specific mechanisms. With BGP on the vRouter agent, BGP can be used to provide a liveness detection mechanism between the tenant on the local compute node and the services that the specific tenant VM is providing.

Configuring BGPaaS using VNC API

To configure BGPaaS using VNC APIs:

1. Access the default project.

   ```
   default_project = self._vnc_lib.project_read(fq_name=[u'default-domain', ‘bgpaas-tenant’])
   ```

2. Create a BGPaaS object.

   ```
   bgpaas_obj = BgpAsAService(name=‘bgpaas_1’, parent_obj=default_project)
   ```

3. Attach the BGP object to a precreated virtual machine interface.

   ```
   bgpaas_obj.add_virtual_machine_interface(vmi)
   ```

4. Set the ASN. It must be an eBGP session.

   ```
   bgpaas_obj.set_autonomous_system('65000')
   ```

   If the ASN is not set, the primary instance IP will be chosen.

   ```
   bgpaas_obj.set_bgpaas_ip_address(u’10.1.1.5’)
   ```

5. Set session attributes.

   ```
   bgp_addr_fams = AddressFamilies(['inet', ‘inet6’]) bgp_sess_attrs =
   BgpSessionAttributes(address_families=bgp_addr_fams,hold_time=60)
   bgpaas_obj.set_bgpaas_session_attributes(bgp_sess_attrs) self._vnc_lib.bgp_as_a_service_create(bgpaas_obj)
   ```

To delete a BGPaaS object, follow the given code:

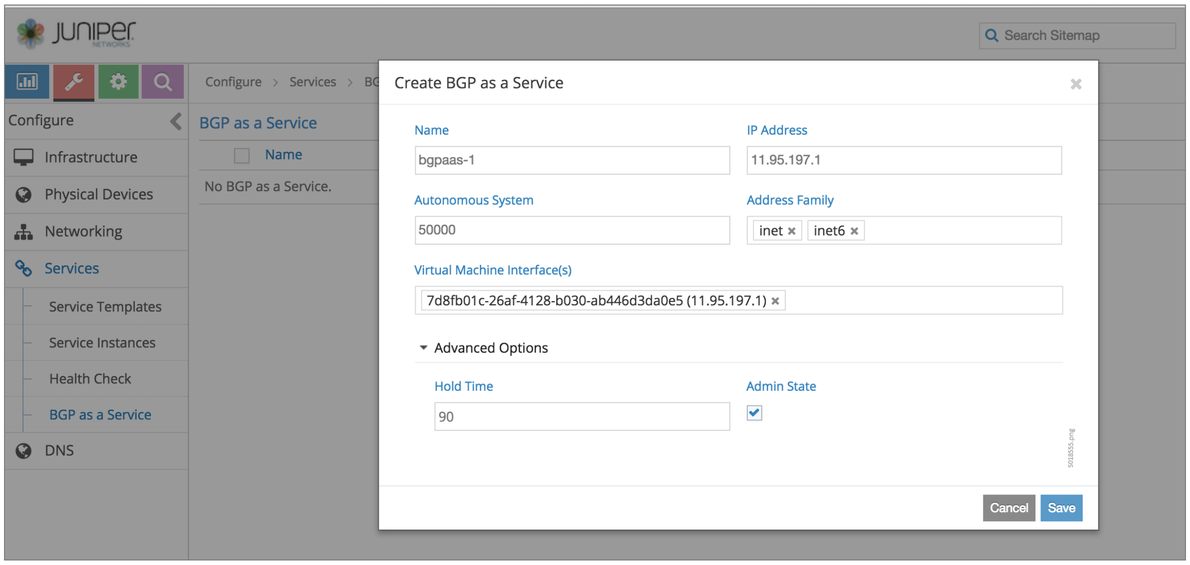
```
fq_name=[u'default-domain', ‘bgpaas-tenant’, ‘bgpaas_1’] bgpaas_obj =
self._vnc_lib.bgp_as_a_service_read(fq_name=fq_name) bgpaas_obj.del_virtual_machine_interface(vmi)
self._vnc_lib.bgp_as_a_service_update(bgpaas_obj) self._vnc_lib.bgp_as_a_service_delete(id=bgpaas_obj.get_uuid())
```

Configuring BGPaaS from Contrail Web UI

To configure BGPaaS within a tenant:

1. Select **Configure** > **Services** > **BGP as a Service** from the Contrail Web User Interface (UI). The BGP as a Service page is displayed.
2. Click the **+** button on the **BGPaaS** page. The **Create BGP as a Service** page is displayed. See Figure 13 on page 101.

Figure 13: Create BGP as a Service



Create BGP as a Service

Name: bgpaas-1

IP Address: 11.95.197.1

Autonomous System: 50000

Address Family: inet, inet6

Virtual Machine Interface(s): 7d8fb01c-26af-4128-b030-ab446d3da0e5 (11.95.197.1)

Advanced Options

Hold Time: 90

Admin State: ✓

3. In the **Create BGPaaS** page, populate the fields with the following values to create your service.

Fields	Description
Name	Enter a name for the BGP service The name can be a unique string of not more than 15 characters that contains alphanumeric characters and hyphen (-).
IP Address	Enter the IPv4 or IPv6 source-address on the BGPaaS VM.
Virtual Machine Interface	Enter IP address of a virtual machine interface.
Address Family	Choose inet or inet6 from the Address Family list according to your requirement.
Autonomous System	Enter AS number in the range 1- 65,534.
Advanced Options	

(Continued)

Fields	Description
Hold Time	Enter the maximum time a BGP session remains active if no Keepalives are received.
Admin State	Select the **Admin state** box to enable the state as UP and deselect the box to disable the state to DOWN.

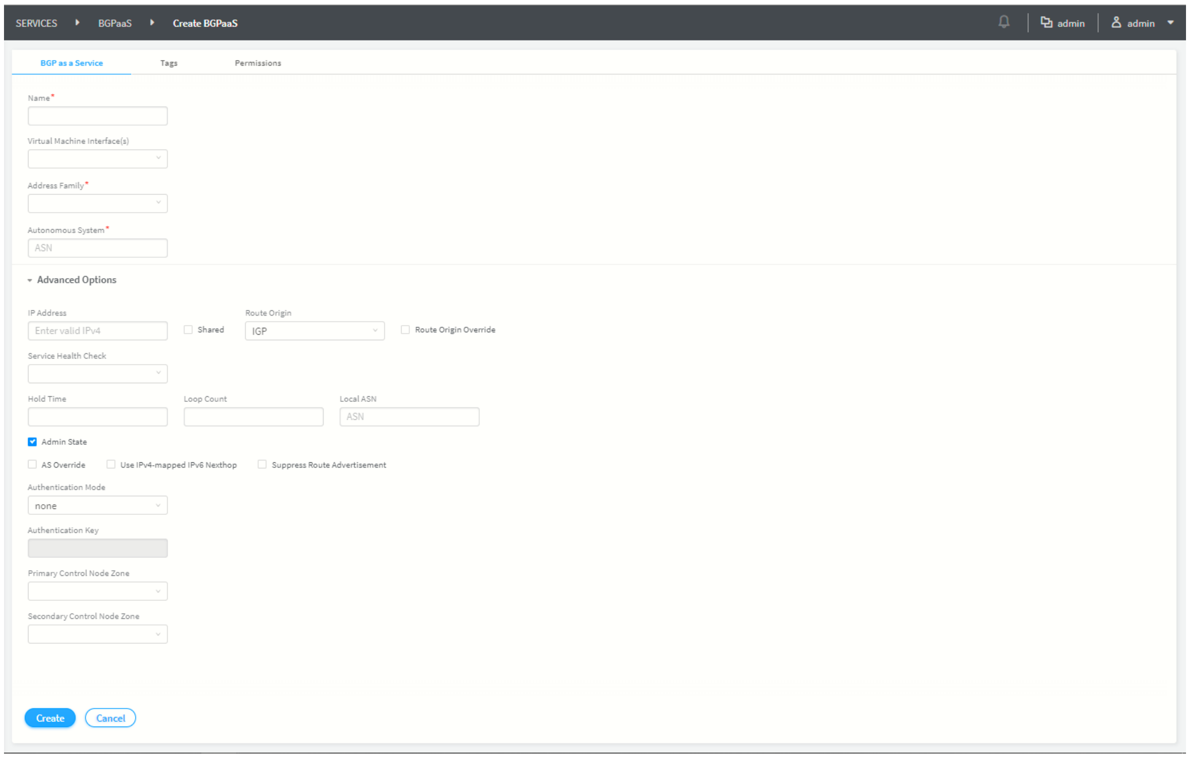
4. Click **Save** to create the BGP object.

Configuring BGPaaS from Contrail Command

To configure BGPaaS within a tenant:

1. Select **Services** > **BGPaaS.** from the Contrail Command user interface (UI). The BGPaaS page is displayed.
2. Click the **Create** button on the **BGPaaS** page. The **Create BGPaaS** page is displayed. See Figure 14 on page 103.

Figure 14: Create BGPaaS



3. In the **Create BGPaaS** page, populate the fields with the following values to create your BGP object.

Fields	Description
Name	Enter a name for the BGP service. The name can be a unique string of not more than 15 characters that contains alphanumeric characters and hyphen (-).
Virtual Machine Interface	Enter IP address of a virtual machine interface.
Address Family	Choose inet or inet6 from the Address Family list according to your requirement.
Autonomous System	Enter autonomous system (AS) number in the range of 1-65,535. If you enable **4 Byte ASN** in **Global Config**, you can enter 4-byte AS number in the range of 1-4,294,967,295.
Advanced Options	

(Continued)

Fields	Description
IP Address	Enter the IPv4 or IPv6 source-address on the BGPaaS VM.
Shared	Select this check box to link all VMIs with the common bgp-router object. If this box is not selected, each virtual machine interface individually links to its own bgp-router object.
Route Origin	• Choose BGP from the list if the route originated on a BGP router. • Choose EGP from the list if the route originated from an External Gateway Protocols (EGP) session. • Choose Incomplete from the list if the Network Layer Reachability Information (NLRI) is learned through methods such as redistribution of the routes into BGP, and not through BGP.
Route Origin Override	Select this check box to override the origin attribute of the advertised route origin into Incomplete.
Service Health Check	Select any Service Health Check object from the list according to your requirement.
Hold Time	Enter the maximum time a BGP session remains active if no Keepalives are received.
Loop Count	Enter the number of times the same ASN can be seen in a route-update. The route is discarded when the loop count is exceeded.
Local ASN	Enter autonomous system (AS) number in the range of 1-65,535. If you enable **4 Byte ASN** in **Global Config**, you can enter 4-byte AS number in the range of 1-4,294,967,295.
AS Override	Select this check box to replace the AS number of the control node with the AS number of the tenant VM.

(Continued)

Fields	Description
Use IPv4-mapped IPv6 Nexthop	Select this check box to use IPv4-mapped IPv6 format as the next hop instead of the IPv4 subnet's default gateway.
Suppress Route Advertisement	Select this check box to prevent advertisement of routes to tenant VM.
Primary Control Node Zone	You can choose the control-node with which the BGPaaS VM can perform a BGP session.
Secondary Control Node Zone	You can choose the control-node with which the BGPaaS VM can perform a BGP session.

4. Click the **Create** button to create the BGP object.

Fat Flows

IN THIS SECTION

Service Providers provide services to several subscribers and as a result, large volume of flows are processed at the Contrail vRouter-level and Contrail Agent-level. Processing large volume of flows affects the flow setup rate and increases latency. Fat flow helps reduce the number of flows that are handled by Contrail.

Understanding Fat Flow

Contrail Networking optimizes the number of flows that are sent or received by a virtual machine by reusing a flow. A single flow pair or a fat flow comprises of a single forward and single reverse flow entry. A fat flow is used for a number of sessions between two end points that use the same application protocol.

For example, multiple DNS sessions from a client to a server can be set up by using a single flow pair. In Contrail Neyworking, the flow key can be reduced from five tuples to two tuples, which consists of source IP address, destination IP address, server port, and internet protocol. This can be configured by specifying the fat flow protocol on the virtual machine interface. The client port, however, is not used in the flow key.

You can configure fat flows by specifying the list of fat-flow protocols on a virtual machine interface. For each such application protocol, the list contains the protocol and port pairs. If you want to enable the fat flow feature on the client side, the configuration must be applied on the client virtual machine interface as well. Contrail Networking also enables you to configure fat flow at the virtual network (VN) level. When configured at the VN level, the fat flow configuration is applied to all VMIs under the configured VN.

Contrail Networking supports aggregation of multiple flows into a single flow by ignoring source and destination ports or IP addresses, with the following possible options:

- ignore source and/or destination ports
- ignore source and/or destination IP addresses
- ignore a combination of source and/or destination ports and IP addresses

Prefix-Based Fat Flow

Contrail Networking enables you to configure the Ignore Address field that reduces the number of flows. You can also create fat flows by configuring prefix length. Service provider subscribers in a common IP address pool can access any IP address in the pool. Contrail Networking also supports prefix-based fat flows. Prefix-based fat flow supports mask processing where you can create flows based on a group of subscribers. This ensures that continuous flows in the same subnet are grouped into a common fat flow that is configured with the same protocol and port numbers. You can apply prefix length-based fat flow on source IP address while the Ignore Address option is configured on the destination IP address, resulting in a reduction of flow processing.

For example, you use prefix-based fat flow to create one flow for 255 IP end points in a /24 subnet (aggregate) mask or one flow for 65,535 IP end points in a /16 subnet (aggregate) mask. This results in a huge reduction on the number of flows created, and a corresponding increase in the number of traffic flows going through vRouter without being limited by vRouter flow setup rate.

Configuring Fat Flow from Contrail Command

You use the Contrail Command user interface (UI) to configure fat flow.

You can configure fat flow from:

- **Overlay>Ports** or
- **Overlay>Virtual Networks**

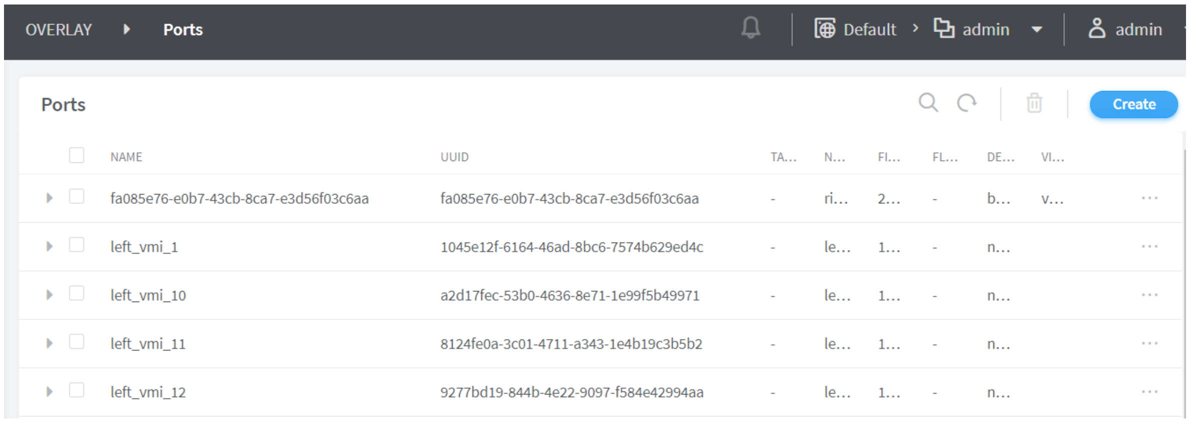
Configuring Fat Flow from Overlay>Ports

To configure fat flow from **Overlay>Ports**:

1. Click **Overlay>Ports**.

   The Ports page is displayed. See Figure 15 on page 107.

Figure 15: Ports Page

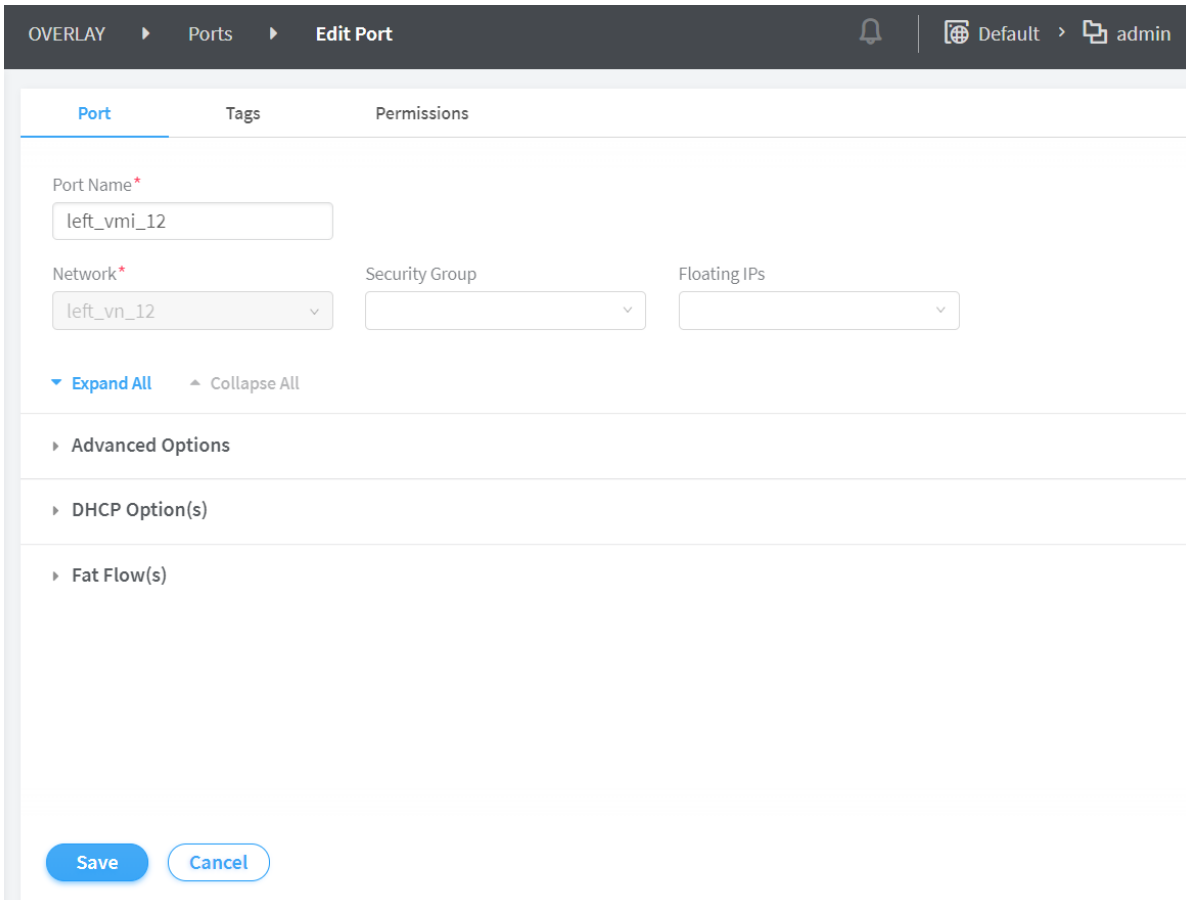


NAME	UUID	TA...	N...	FI...	FL...	DE...	VI...
fa085e76-e0b7-43cb-8ca7-e3d56f03c6aa	fa085e76-e0b7-43cb-8ca7-e3d56f03c6aa	-	ri...	2...	-	b...	v...
left_vmi_1	1045e12f-6164-46ad-8bc6-7574b629ed4c	-	le...	1...	-	n...	
left_vmi_10	a2d17fec-53b0-4636-8e71-1e99f5b49971	-	le...	1...	-	n...	
left_vmi_11	8124fe0a-3c01-4711-a343-1e4b19c3b5b2	-	le...	1...	-	n...	
left_vmi_12	9277bd19-844b-4e22-9097-f584e42994aa	-	le...	1...	-	n...	

2. Select the port you want to configure by selecting the check box next to the name of the port, and then click the **Edit** icon.

   The Edit Port page is displayed. See Figure 16 on page 108.

Figure 16: Edit Port Page


OVERLAY ▸ Ports ▸ Edit Port

Default › admin

Port | Tags | Permissions

Port Name*
left_vmi_12

Network*
left_vn_12

Security Group

Floating IPs

Expand All · Collapse All

Advanced Options

DHCP Option(s)

Fat Flow(s)

Save · Cancel

3. Click **Fat Flow(s)** to display the fields that you can edit.

   You can edit the fields listed in Table 8 on page 109.

Table 8: Edit Fat Flow(s)

Field	Action
Protocol	Change the protocol that is currently being used to any one of the following protocols given in the **Protocol** list: • ICMP • SCTP • TCP (default) • UDP You can select ICMP for both IPv4 and IPv6 traffic.
Port	Edit the Port field to any value between 0 through 65,535. Enter 0 to ignore both source and destination port numbers. **NOTE**: If you select ICMP as the protocol, the **PORT** field is not enabled.

Table 8: Edit Fat Flow(s) ***(Continued)***

Field		Action
Ignore Address		Change the Ignore Address field to any one of the following options: • **Destination**—If you choose Destination as the option, Prefix Aggregation Source fields are only enabled. See Figure 17 on page 112. • **None** (default)—If you choose None as the option, both Prefix Aggregation Source and Prefix Aggregation Destination fields are enabled. See Figure 18 on page 113. • **Source**—If you choose Source as the option, Prefix Aggregation Destination fields are only enabled. See Figure 19 on page 113. **NOTE**: Fat flow in Contrail Networking supports aggregation of multiple flows into a single flow by ignoring source and destination ports or IP addresses.
Prefix Aggregation Source	**Source Subnet**	Edit source IP subnet. Ensure that the source subnet of the flows match. For example, to create fat flows with 192.0.2.0/24 as the subnet, enter 192.0.2.0/24 in the **Source Subnet** field. Valid range of the subnet mask: /8 through /32. For more information, refer to the **Understanding Source and Destination** section.

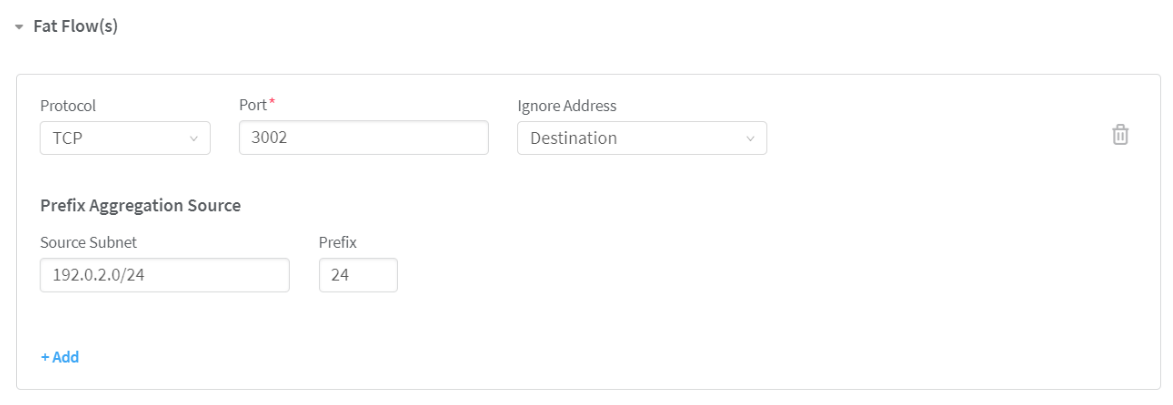
Table 8: Edit Fat Flow(s) *(Continued)*

Field		Action
	Prefix	Edit source subnet prefix length. The prefix length you enter is used to aggregate flows matching the source subnet. For example, when the source subnet is 10.1.0.0/16 and prefix length is 24, the flows matching the source subnet is aggregated to 10.1.x.0/24 flows. Valid range of the prefix length: /(subnet mask of the source subnet) through /32. For more information, refer to the **Understanding Source and Destination** section. **NOTE**: Contrail Networking enables you to configure subnet and prefix length.
Prefix Aggregation Destination	**Destination Subnet**	Edit destination IP address. Ensure that the destination subnet of the flows match. For example, to create fat flows with 192.0.2.0/24 as the subnet, enter 192.0.2.0/24 in the **Destination Subnet** field. Valid range of the subnet mask: /8 through /32. For more information, refer to the **Understanding Source and Destination** section.

Table 8: Edit Fat Flow(s) *(Continued)*

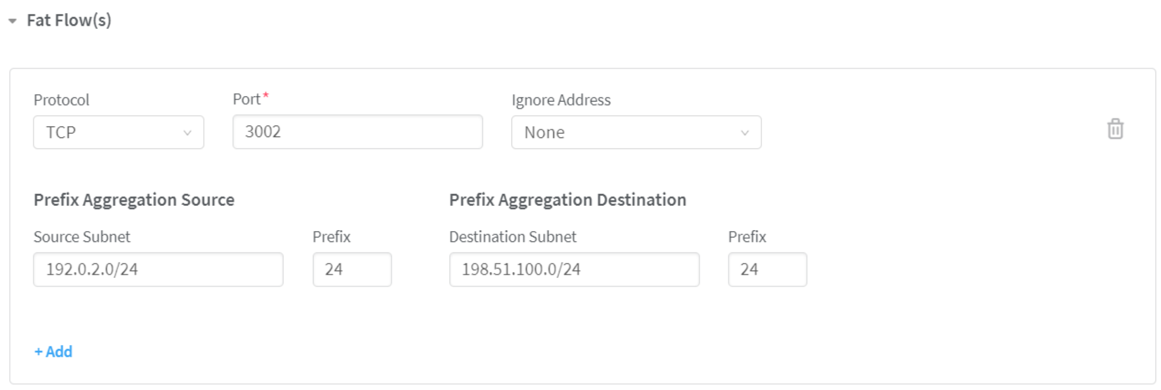
Field		Action
	Prefix	Edit destination subnet prefix length. The prefix length you enter is used to aggregate flows matching the destination subnet. For example, when the destination subnet is 10.1.0.0/16 and prefix length is 24, the flows matching the destination subnet is aggregated to 10.1.x.0/24 flows. Valid range of the prefix length: /(subnet mask of the destination subnet) through /32. For more information, refer to the **Understanding Source and Destination** section. **NOTE**: Contrail Networking enables you to configure subnet and prefix length.

Figure 17: Ignore Address—Destination



Protocol: TCP
Port*: 3002
Ignore Address: Destination
Source Subnet: 192.0.2.0/24
Prefix: 24

Figure 18: Ignore Address—None


Fat Flow(s)

Protocol	Port*	Ignore Address
TCP	3002	None

Prefix Aggregation Source

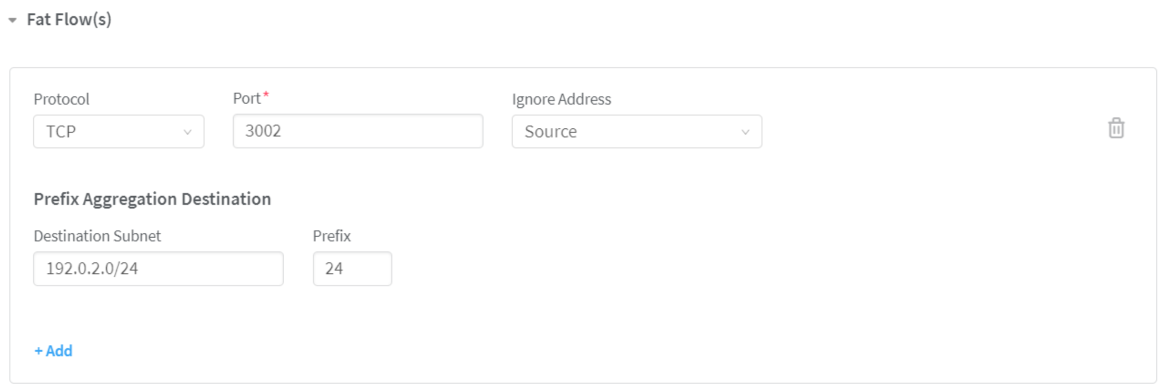
Source Subnet	Prefix
192.0.2.0/24	24

Prefix Aggregation Destination

Destination Subnet	Prefix
198.51.100.0/24	24

+ Add

Figure 19: Ignore Address—Source


Fat Flow(s)

Protocol	Port*	Ignore Address
TCP	3002	Source

Prefix Aggregation Destination

Destination Subnet	Prefix
192.0.2.0/24	24

+ Add

4. Click **Save** to update new configuration information.

NOTE: **Understanding Source and Destination**

- **Source**—For packets from the local virtual machine, source refers to the source IP of the packet. For packets from the physical interface, source refers to the destination IP of the packet.
- **Destination**—For packets from the local virtual machine, destination refers to the destination IP of the packet. For packets from the physical interface, destination refers to the source IP of the packet.

Configuring Fat Flow from Overlay>Virtual Networks

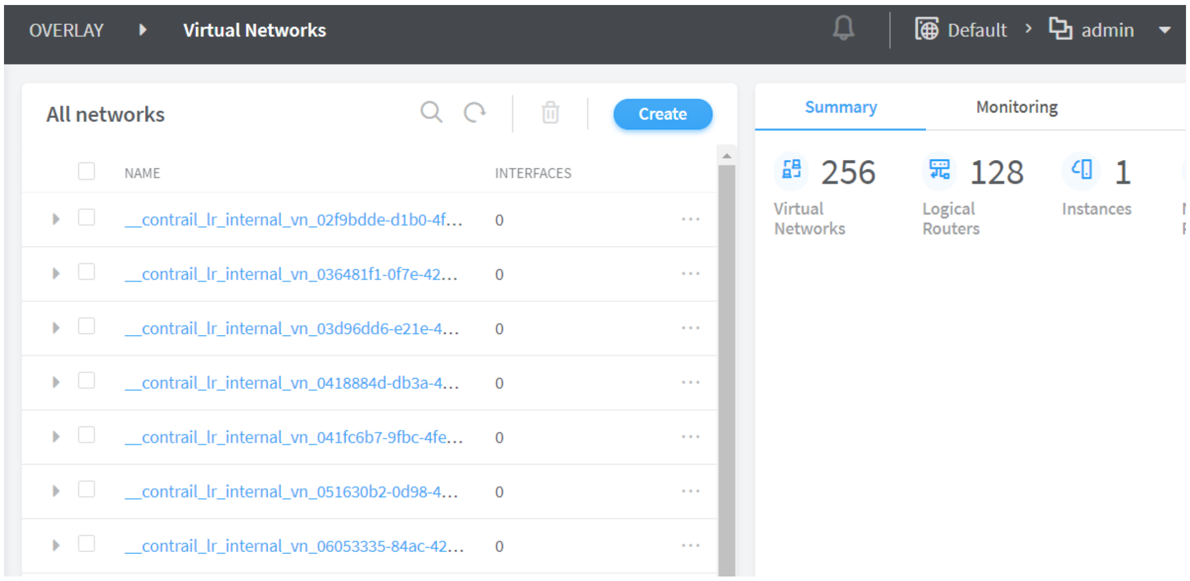
Contrail Networking also enables you to configure fat flow at the virtual network (VN) level. When you configure fat flow from the VN level, the fat flow configuration is applied to all VMIs under the configured VN.

To configure fat flow from **Overlay>Virtual Networks**:

1. Click **Overlay>Virtual Networks**.

   The Virtual Networks page is displayed. See Figure 20 on page 114.

Figure 20: Virtual Networks Page

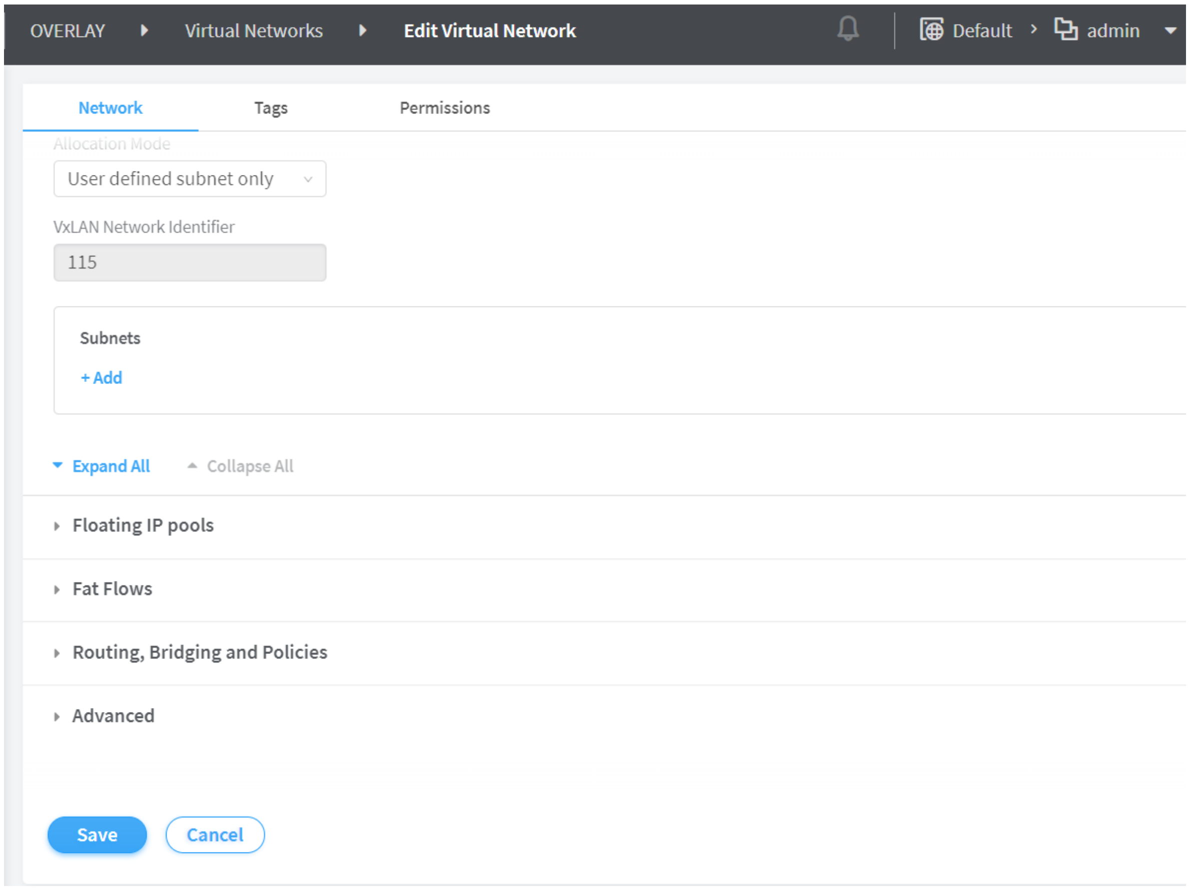


NAME	INTERFACES
__contrail_lr_internal_vn_02f9bdde-d1b0-4f...	0
__contrail_lr_internal_vn_036481f1-0f7e-42...	0
__contrail_lr_internal_vn_03d96dd6-e21e-4...	0
__contrail_lr_internal_vn_0418884d-db3a-4...	0
__contrail_lr_internal_vn_041fc6b7-9fbc-4fe...	0
__contrail_lr_internal_vn_051630b2-0d98-4...	0
__contrail_lr_internal_vn_06053335-84ac-42...	0

2. Select the virtual network you want to edit by selecting the check box next to the name of the virtual network, and then click the **Edit** icon.

   The Edit Virtual Network page is displayed. See Figure 21 on page 115.

Figure 21: Edit Virtual Network Page



Allocation Mode: User defined subnet only

VxLAN Network Identifier: 115

3. Click **Fat Flows** to display the fields that you can edit.

   You can edit the fields listed in Table 9 on page 116.

Table 9: Edit Fat Flows

Field	Action
Protocol	Change the protocol that is currently being used to any one of the following protocols given in the **Protocol** list: • ICMP • SCTP • TCP (default) • UDP You can select ICMP for both IPv4 and IPv6 traffic.
Port	Edit the Port field to any value between 0 through 65,535. Enter 0 to ignore both source and destination port numbers. **NOTE**: If you select ICMP as the protocol, the **PORT** field is not enabled.

Table 9: Edit Fat Flows *(Continued)*

Field		Action
Ignore Address		Change the Ignore Address field to any one of the following options: • **Destination**—If you choose Destination as the option, Prefix Aggregation Source fields are only enabled. See Figure 17 on page 112. • **None** (default)—If you choose None as the option, both Prefix Aggregation Source and Prefix Aggregation Destination fields are enabled. See Figure 18 on page 113. • **Source**—If you choose Source as the option, Prefix Aggregation Destination fields are only enabled. See Figure 19 on page 113. **NOTE**: Fat flow in Contrail Networking supports aggregation of multiple flows into a single flow by ignoring source and destination ports or IP addresses.
Prefix Aggregation Source	**Source Subnet**	Edit source IP address. Ensure that the source subnet of the flows match. For example, to create fat flows with 192.0.2.0/24 as the subnet, enter 192.0.2.0/24 in the **Source Subnet** field. Valid range of the subnet mask: /8 through /32. For more information, refer to the **Understanding Source and Destination** section.

Table 9: Edit Fat Flows *(Continued)*

Field		Action
	Prefix	Edit source subnet prefix length. The prefix length you enter is used to aggregate flows matching the source subnet. For example, when the source subnet is 10.1.0.0/16 and prefix length is 24, the flows matching the source subnet is aggregated to 10.1.x.0/24 flows. Valid range of the prefix length: /(subnet mask of the source subnet) through /32. For more information, refer to the **Understanding Source and Destination** section. **NOTE**: Contrail Networking enables you to configure subnet and prefix length.
Prefix Aggregation Destination	**Destination Subnet**	Edit destination IP address. Ensure that the destination subnet of the flows match. For example, to create fat flows with 192.0.2.0/24 as the subnet, enter 192.0.2.0/24 in the **Destination Subnet** field. Valid range of the subnet mask: /8 through /32. For more information, refer to the **Understanding Source and Destination** section.

Table 9: Edit Fat Flows *(Continued)*

Field		Action
	Prefix	Edit destination subnet prefix length. The prefix length you enter is used to aggregate flows matching the destination subnet. For example, when the destination subnet is 10.1.0.0/16 and prefix length is 24, the flows matching the destination subnet is aggregated to 10.1.x.0/24 flows. Valid range of the prefix length: /(subnet mask of the destination subnet) through /32. For more information, refer to the **Understanding Source and Destination** section. **NOTE**: Contrail Networking enables you to configure subnet and prefix length.

4. (Optional) If you have not already added fat flow information, you can add information by clicking **+Add**. You can enter information as given in Table 9 on page 116.

5. Click **Save** to add new configuration information.

NOTE:

- A service virtual machine (SVM) is a virtualized network function (VNF) that is a part of a service chain. Fat flow configuration on SVM is supported when:
  - Left virtual machine interface: Ignore source address and/or Prefix aggregation destination
  - Right virtual machine interface: Ignore destination address and/or Prefix aggregation source
- Fat flow on service virtual machine interfaces (SVMIs) in scale-out mode is supported when all SVMIs are on the same compute, and not on the source or destination compute.
- Fat flow configuration across all SVMs must be consistent.

Limitations of Fat Flow

The following are the limitations of fat flow.

- Drop in packet per second (pps) performance depends on the number of rules or configuration.
- Network policy configuration must be consistent with fat flow configuration.

Use Case: Configuring Fat Flows from Contrail Command

IN THIS SECTION

This topic provides step-by-step instructions to create an in-network service chain and configure fat flows.

A service chain is a set of services that are connected across networks. A service chain consists of service instances, left and right virtual networks, and a service policy attached to the networks. In an in-network service chain, packets are routed between service instance interfaces. When a packet is routed through the service chain, the source address of the packet entering the left interface of the service chain and source address of the packet exiting the right interface is the same. For more information, see Service Chaining. You can also configure fat flows while you create an in-network-NAT or transparent service chain.

Overview

IN THIS SECTION

Service Providers provide services to several subscribers and as a result, large volume of flows are processed at the Contrail vRouter-level and Contrail Agent-level. Processing large volume of flows affects the flow setup rate and increases latency. Fat flow helps reduce the number of flows that are handled by Contrail.

Contrail Networking enables you to configure the Ignore Address field that reduces the number of flows. You can also create fat flows by configuring prefix length. Service provider subscribers in a common IP address pool can access any IP address in the pool. Contrail Networking also supports prefix-based fat flows. Prefix-based fat flow supports mask processing, where you can create flows based on a group of subscribers. This ensures that continuous flows in the same subnet are grouped into a common fat flow that is configured with the same protocol and port numbers. You can apply prefix length-based fat flow on source IP address while the Ignore Address option is configured on the destination IP address, resulting in a reduction of flow processing.

Topology Information

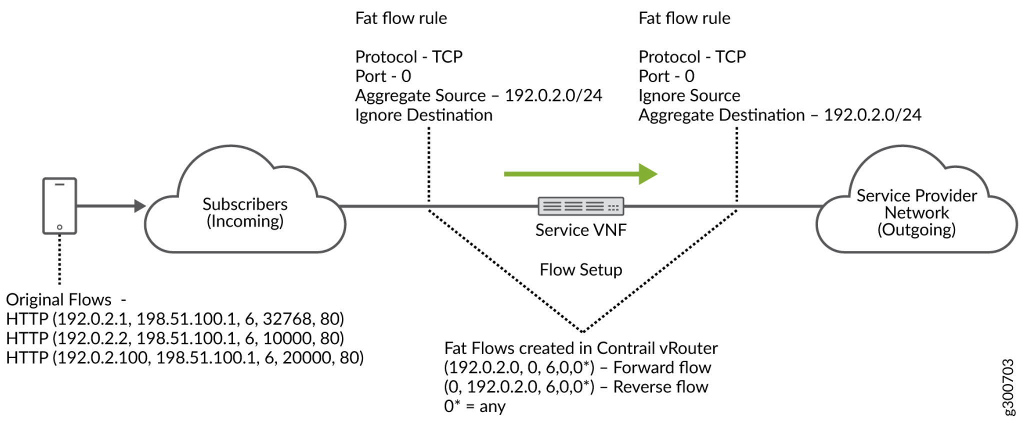
These topologies provide information on how you can configure the Ignore Address field to reduce the number of flows.

Ignore Address - Source, Destination

Figure 22 on page 122 depicts a scenario where you have selected the following options from the **Ignore Address** list.

- **Destination**—for the test-left-VN (subscribers network).
- **Source**—for the test-right-VN (service provider network).

Figure 22: Ignore Source, Destination


Fat flow rule
Protocol - TCP
Port - 0
Aggregate Source – 192.0.2.0/24
Ignore Destination
Fat flow rule
Protocol - TCP
Port - 0
Ignore Source
Aggregate Destination – 192.0.2.0/24
Subscribers (Incoming)
Service VNF
Flow Setup
Service Provider Network (Outgoing)
Original Flows -
HTTP (192.0.2.1, 198.51.100.1, 6, 32768, 80)
HTTP (192.0.2.2, 198.51.100.1, 6, 10000, 80)
HTTP (192.0.2.100, 198.51.100.1, 6, 20000, 80)
Fat Flows created in Contrail vRouter
(192.0.2.0, 0, 6,0,0*) – Forward flow
(0, 192.0.2.0, 6,0,0*) – Reverse flow
0* = any
g300703

Understanding Source and Destination

- **Source**—For packets from the local virtual machine, source refers to the source IP of the packet.
- **Destination**—For packets from the local virtual machine, destination refers to the destination IP of the packet.

By choosing **Destination** in the subscribers network, the Prefix Aggregation Source fields are enabled in the network. And by choosing **Source** in the service providers network, the Prefix Aggregation Destination fields are enabled in the network. When you configure Ignore Address, Contrail Networking helps you to aggregate multiple flows into a single flow by ignoring source and/or destination ports.

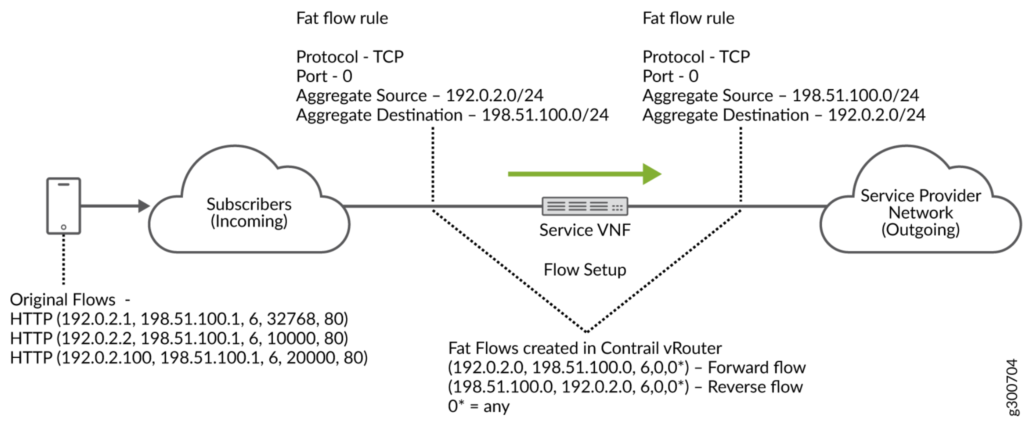
To create fat flows in subscribers network with 192.0.2.0/24 as the subnet, enter 192.0.2.0/24 in the **Source Subnet** field and 24 in the **Prefix** field. The prefix length, 24, is used to aggregate flows matching the source subnet. The flows matching the source subnet is aggregated to 192.0.2.X/24 flows.

Similarly to create fat flows in service provider network with 192.0.2.0/24 as the subnet, enter 192.0.2.0/24 in the **Destination Subnet** field and 24 in the **Prefix** field. The prefix length, 24, is used to aggregate flows matching the destination subnet. The flows matching the destination subnet is aggregated to 192.0.2.X/24 flows.

Ignore Address - None

Figure 23 on page 123 depicts a scenario where you have selected **None** from the **Ignore Address** list.

Figure 23: Ignore None


Fat flow rule
Protocol - TCP
Port - 0
Aggregate Source – 192.0.2.0/24
Aggregate Destination – 198.51.100.0/24
Fat flow rule
Protocol - TCP
Port - 0
Aggregate Source – 198.51.100.0/24
Aggregate Destination – 192.0.2.0/24
Subscribers (Incoming)
Service VNF
Flow Setup
Service Provider Network (Outgoing)
Original Flows -
HTTP (192.0.2.1, 198.51.100.1, 6, 32768, 80)
HTTP (192.0.2.2, 198.51.100.1, 6, 10000, 80)
HTTP (192.0.2.100, 198.51.100.1, 6, 20000, 80)
Fat Flows created in Contrail vRouter
(192.0.2.0, 198.51.100.0, 6,0,0*) – Forward flow
(198.51.100.0, 192.0.2.0, 6,0,0*) – Reverse flow
0* = any
g300704

By choosing **None** in the subscribers network and service providers network, the Prefix Aggregation Destination fields and Prefix Aggregation Source fields are enabled in both networks.

In this scenario, the subnet that you enter in the Source Subnet field of the subscribers network matches the subnet that you enter in Destination Subnet field of the service providers network. Similarly, the subnet that you enter in the Destination Subnet field of the subscribers network matches the subnet that you enter in the Source Subnet field of the service providers network.

Prerequisites

Before you begin, ensure that the following prerequisites are met.

- **Hardware Requirements**
  - Processor: 4 core x86
  - Memory: 32GB RAM
  - Storage: at least 128GB hard disk
- **Software Requirements**
  - Contrail Networking Release 5.0 or later
- Create three network IPAMs (IP Address Management).

  You can create a new Network IPAM by following these steps:

  1. Click **Overlay**>**IPAM**.

The IP Address Management page is displayed.

2. Click **Create** to create a new network IPAM.
3. In the **Name** field, enter a name for the IPAM.

   For left network, enter **test-left-IPAM**. For right network, enter **test-right-IPAM**. For management network, enter **mgmt-right-IPAM**.
4. Select **Default** from the DNS list.
5. Enter valid IP address in the **NTP Server IP** field.
6. Enter domain name in the **Domain Name** field.
7. Click **Create**.

   The IP Address Management page is displayed.

Getting Started

The instructions provided in the topics given below will help you to

1. Create the following virtual networks:
   - Left Virtual Network
   - Right Virtual Network
   - Management Virtual Network

   For steps to create virtual networks, see "Create Virtual Network" on page 126.
2. Create three virtual machines.

   Each virtual machine must be created with left, right, and management interfaces.
   - Left Virtual Machine
   - Right Virtual Machine
   - Management Virtual Machine

   For steps to create virtual machines by using OpenStack, see "Create Virtual Machines by using OpenStack" on page 127.

   For steps to create virtual machines by using Contrail Command, see "Create Virtual Machines by using Contrail Command" on page 128.

3. Create a service template.

   For steps to create a service template, see "Create Service Template" on page 129.

4. Add a service instance.

   For steps to add a service instance, see "Add Service Instance" on page 130.

5. Configure fat flows for these virtual networks.

   - Left Virtual Network
   - Right Virtual Network

   For steps to configure fat flows, see "Configure Fat Flow" on page 131.

6. Create a service policy for the left virtual network and right virtual network.

   For steps to create a service policy, see "Create Service Policy" on page 134.

7. Attach the service policy to the left virtual network and right virtual network.

   For steps to attach a service policy to a virtual network, see "Attach Service Policy" on page 135.

8. Ping right virtual machine from left virtual machine.

   For steps to ping the right virtual machine by using OpenStack, see "Launch a Virtual Machine from OpenStack" on page 135.

   For steps to ping the right virtual machine by using Contrail Command, see "Launch a Virtual Machine from Contrail Command" on page 136.

Configuration

IN THIS SECTION

These topics provide instructions to configure fat flows by creating an in-network service chain.

Create Virtual Network

Use the Contrail Command UI to create a left virtual network, right virtual network, and management virtual network.

To create a left virtual network:

1. Click **Overlay**>**Virtual Networks**.

   The All Networks page is displayed.
2. Click **Create** to create a network.

   The Create Virtual Network page is displayed.
3. In the **Name** field enter **test-left-VN** for the left virtual network.
4. Select **(Default) User defined subnet only** from the **Allocation Mode** list.
5. Click **+Add** in the Subnets section to add subnets.

   In the row that is displayed,

   a. Click the arrow in the Network IPAM field and select **left-ipam** for the left virtual network.

      For the right virtual network, select **right-ipam** and for the management network, select **mgmt-ipam**.

      NOTE: Management network is not used to route packets. This network is used to help debug issues with the virtual machine.
6. Enter **192.0.2.0/24** in the **CIDR** field.
7. Click **Create**.

   The All Networks page is displayed. All virtual networks that you created are displayed in this page.

Repeat steps 2 through 7 to create the right virtual network (**test-right-VN**) and management virtual network (**test-mgmt-VN**).

Create Virtual Machine

IN THIS SECTION

You use OpenStack or Contrail Command to create virtual machines for left, right, and management networks. You create the virtual networks with left, right, and management interfaces.

Create Virtual Machines by using OpenStack

Follow these steps to create left virtual machine by using OpenStack.

1. Click **Project**>**Compute**>**Instances**.

   The Instances page is displayed.
2. Click **Launch Instance** to create an instance.

   The Details tab of the Launch Instance page is displayed.
3. Enter **test-left-VM** for the left virtual machine in the **Instance Name** field and click the **Source** tab.

   The Source tab of the Launch Instance page is displayed.
4. Select an **vSRX** image from the Available list by clicking the add **(+)** icon next to the image file.
5. Click the **Flavor** tab.

   The Flavor tab of the Launch Instance page is displayed.

   NOTE: vSRX image with M1.large flavor is recommended for in-network virtual machine.
6. Select **M1.large** as the flavor from the Available list by clicking the add **(+)** icon next to the flavor name.
7. Click the **Networks** tab.

   The Network tab of the Launch Instance page is displayed.
8. Select a network you want to associate with the virtual machine instance by clicking the add **(+)** icon next to the network name.

   For the left virtual machine, select **test-left-VN**. For the right virtual machine, select **test-right-VN**. For the management virtual machine, select **test-mgmt-VN**.
9. Click **Launch Instance** to launch the virtual machine instance.

   The Instances page is displayed.

All virtual machine instances that you created are displayed on the Instances page.

Repeat steps 2 through 9 to create the right virtual machine (**test-right-VM**) and management virtual machine (**test-mgmt-VM**).

Create Virtual Machines by using Contrail Command

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

1. Click **Workloads** > **Instances**.

   The Instances page is displayed.
2. Click **Create**.

   The Create Instance page is displayed.
3. Select **Virtual Machine** option button as the serve type.
4. Enter **test-left-VM** for the left virtual machine in the **Instance Name** field.
5. Select **Image** as the boot source from the **Select Boot Source** list.

   NOTE: vSRX image with M1.large flavor is recommended for in-network virtual machine.
6. Select **vSRX image** file from the **Select Image** list.
7. Select **M1.large** flavor from the **Select Flavor** list.
8. Select the network you want to associate with the left virtual machine by clicking **>** next to the name of the virtual machine listed in the Available Networks table.

   For the left virtual machine, select **test-left-VN**. For the right virtual machine, select **test-right-VN**. For the management virtual machine, select **test-mgmt-VN**.

   The network is added to the Allocated Networks table.
9. Select **nova** from the **Availability Zone** list.

   NOTE: You can choose any other availability zone.
10. Select **5** from the **Count (1-10)** list.

    NOTE: You can choose any value from 1 through 10.
11. Click **Create** to launch the left virtual machine instance.

    The Instances page is displayed. The virtual machine instances that you created are listed on the Instances page.

Repeat steps 2 through 11 to create right virtual machine instance (**test-right-VM**) and management virtual machine instance (**test-mgmt-VM**).

Create Service Template

Follow these steps to create a service template by using the Contrail Command UI:

1. Click **Services**>**Catalog**.

   The VNF Service Templates page is displayed.
2. Click **Create**.

   The Create VNF Service Template page is displayed.
3. Enter **test-service-template** in the **Name** field.
4. Select **v2** as the version type.

   NOTE: Contrail Networking supports only *Service Chain Version 2* (**v2**).
5. Select **Virtual Machine** as the virtualization type.
6. Select **In-Network** as the service mode.
7. Select **Firewall** as the service type.
8. From the Interface section,

   - Select **left** as the interface type from the **Interface Type** list.
   - Click **+ Add**.

     The Interface Type list is added to the table.

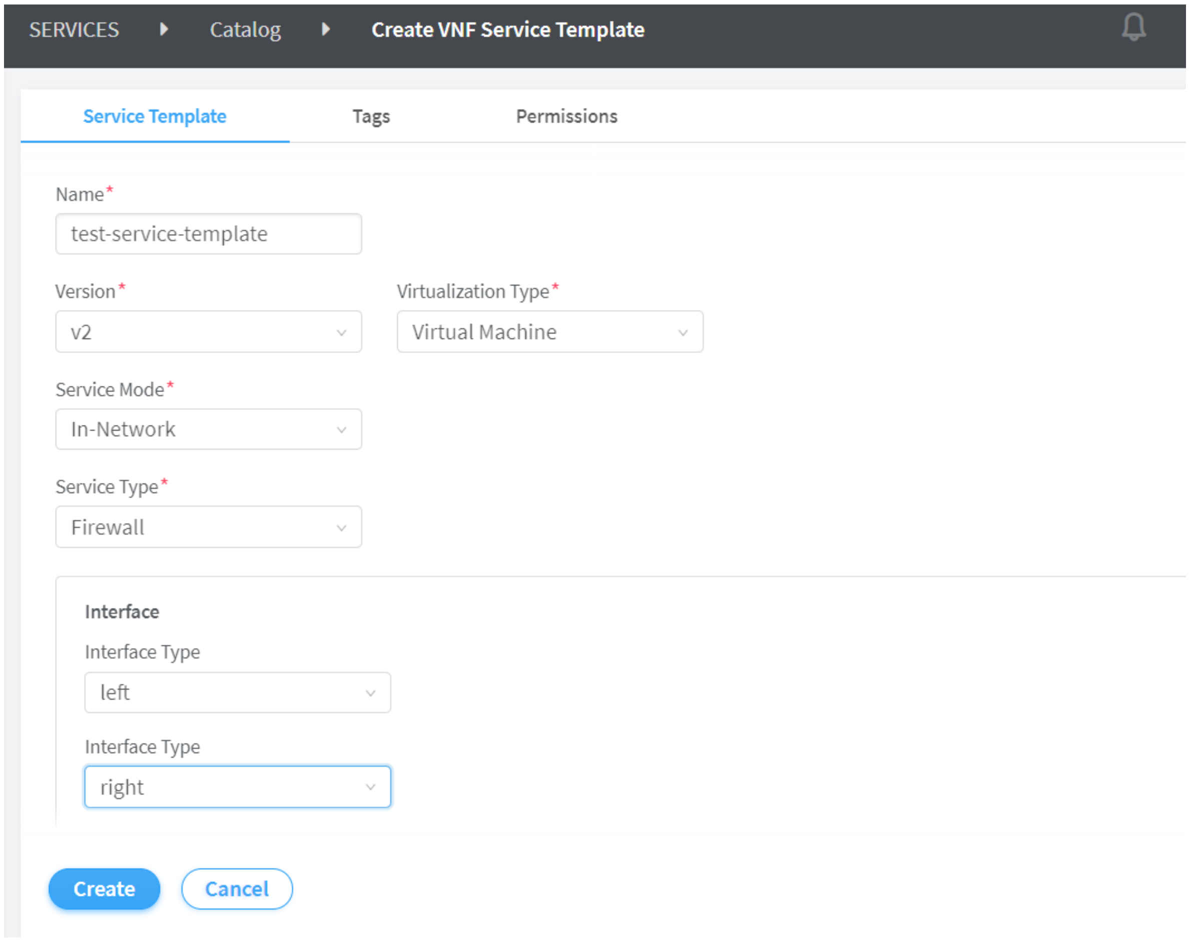
     Select **right** as the interface type.
   - Click **+ Add** again.

     Another Interface Type list is added to the table.

     Select **management** as the interface type.

   NOTE: The interfaces created on the virtual machine must follow the same sequence as that of the interfaces in the service template.

Figure 24: Adding Interfaces



Name* test-service-template

Version* v2

Virtualization Type* Virtual Machine

Service Mode* In-Network

Service Type* Firewall

Interface

Interface Type left

Interface Type right

9. Click **Create** to create the service template.

   The VNF Service Templates page is displayed. The service template that you created is displayed in the VNF Service Templates page.

Add Service Instance

Follow these steps to add a service instance by using the Contrail Command UI:

1. Click **Services**>**Deployments**.

   The VNF Service Instances page is displayed.

2. Click **Create**.

   The Create VNF Service Instance page is displayed.

3. Enter **test-service-instance** in the **Name** field.

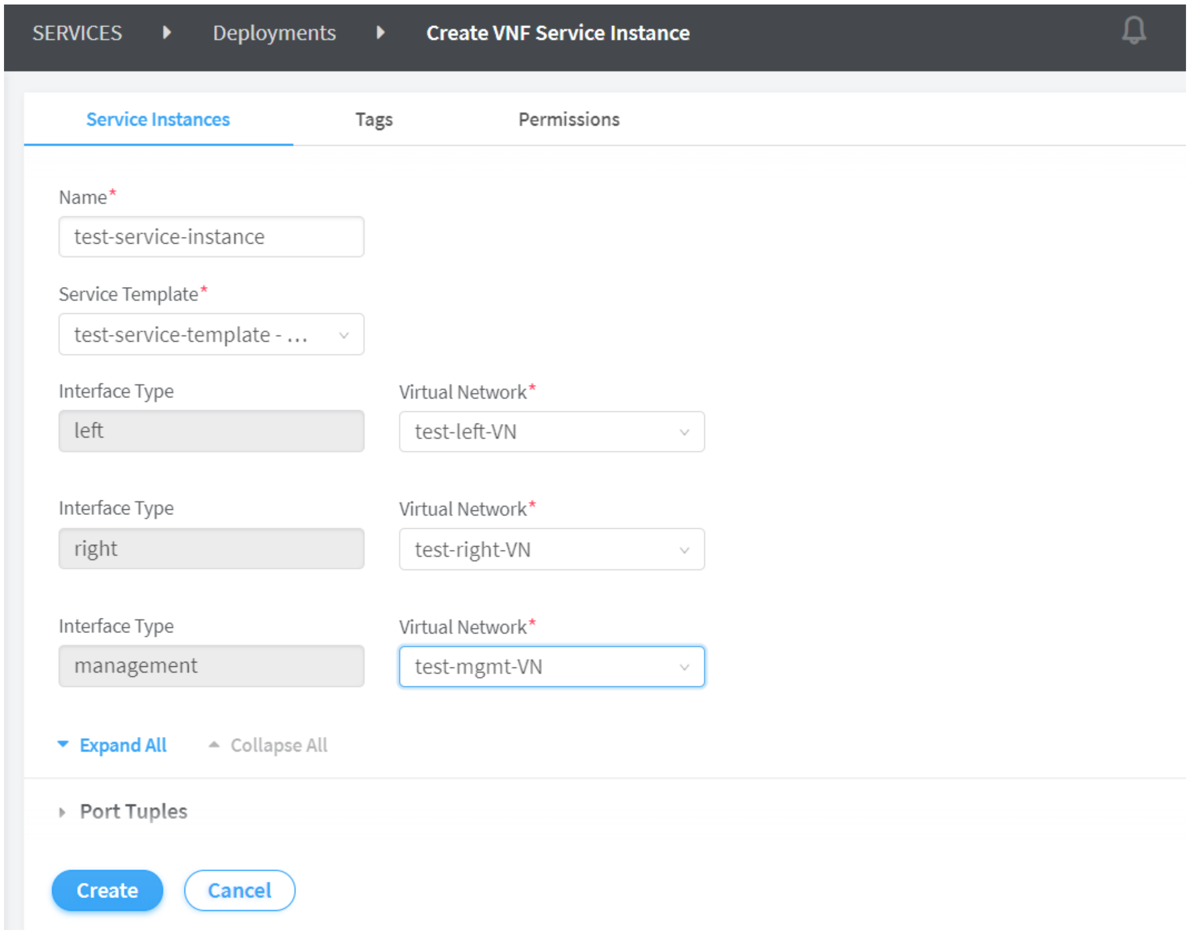
4. Select **test-service-template - [in-network, (left, right, management)] - v2** from the **Service Template** list.

   The **Interface Type** and **Virtual Network** fields are displayed.

5. Select the virtual network for each interface type as given below.

   - **left**—Select the left virtual network (**test-left-VN**) that you created.
   - **right**—Select the right virtual network (**test-right-VN**) that you created.
   - **management**—Select the management virtual network (**test-management-VN**) that you created.

Figure 25: Adding Service Instance



Name* test-service-instance

Service Template* test-service-template - ...

Interface Type: left	Virtual Network*: test-left-VN
Interface Type: right	Virtual Network*: test-right-VN
Interface Type: management	Virtual Network*: test-mgmt-VN

6. Click **Create** to create the service instance.

   The VNF Service Instances page is displayed. The service instance that you created is displayed in the VNF Service Instances page.

Configure Fat Flow

In Contrail Networking, you can configure fat flow at the virtual network (VN) level. When you configure fat flow from the VN level, the fat flow configuration is applied to all VMIs under the configured VN.

For more information, see "Fat Flows" on page 105.

Follow these steps to configure fat flows by using the Contrail Command UI.

1. Click **Overlay**>**Virtual Networks**.

   The Virtual Networks page is displayed.
2. Select **test-left-VN** by selecting the check box next to the name of the virtual network, and then click the **Edit** icon.

   The Edit Virtual Network page is displayed.

   NOTE: You must configure fat flows on all the virtual networks that you created.
3. Click **Fat Flow(s)** to display the fields that you can edit.

   You can edit the fields listed in Table 10 on page 132.

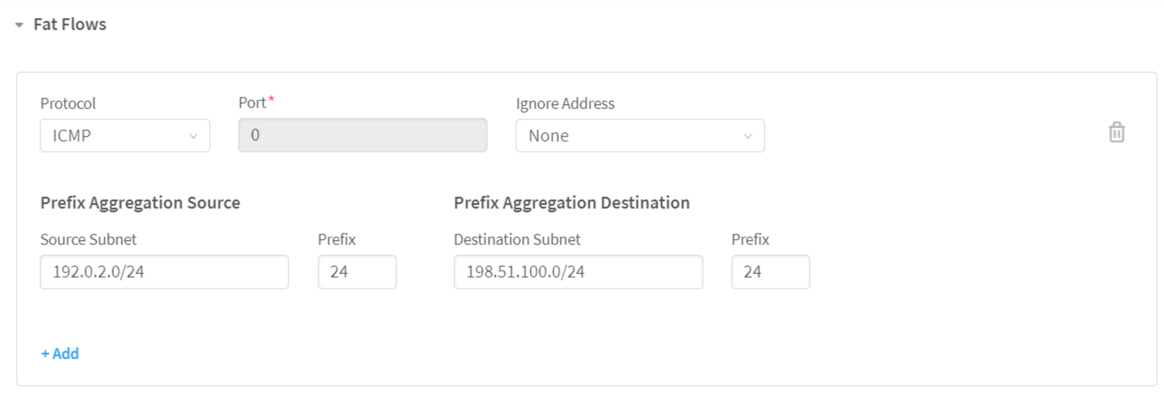
Table 10: Edit Fat Flow(s)

Field	Action
Protocol	Select **ICMP** from the **Protocol** list. You can select ICMP for both IPv4 and IPv6 traffic.
Port	Edit the Port field to any value between 0 through 65,535. Enter 0 to ignore both source and destination port numbers. **NOTE**: If you select ICMP as the protocol, the **PORT** field is not enabled.
Ignore Address	Select **None** from the **Ignore Address** list. For more information on **Destination** and **Source** options, see "Fat Flows" on page 105. **NOTE**: Fat flow in Contrail Networking supports aggregation of multiple flows into a single flow by ignoring source and destination ports or IP addresses.

Table 10: Edit Fat Flow(s) *(Continued)*

Field		Action
Prefix Aggregation Source	Source Subnet	For **test-left-VN**, enter **192.0.2.0/24** in the **Source Subnet** field. See Figure 26 on page 133. For **test-right-VN**, enter **198.51.100.0/24** in the **Source Subnet** field. See Figure 27 on page 134.
	Prefix	Enter **24** in the **Prefix** field.
Prefix Aggregation Destination	Destination Subnet	For **test-left-VN**, enter **198.51.100.0/24** in the **Source Subnet** field. See Figure 26 on page 133. For **test-right-VN**, enter **192.0.2.0/24** in the **Source Subnet** field. See Figure 27 on page 134.
	Prefix	Enter **24** in the **Prefix** field.

Figure 26: Configure Fat Flows for test-left-VN

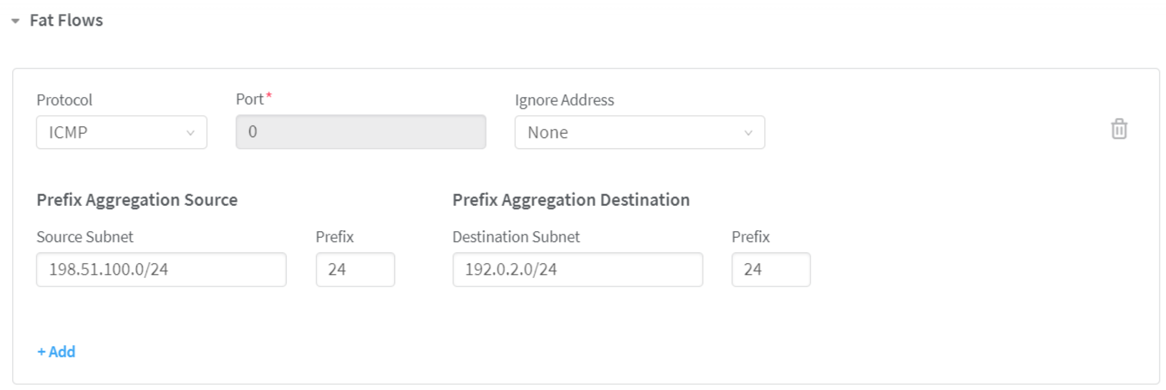


Protocol: ICMP
Port*: 0
Ignore Address: None

Prefix Aggregation Source
Source Subnet: 192.0.2.0/24
Prefix: 24

Prefix Aggregation Destination
Destination Subnet: 198.51.100.0/24
Prefix: 24

Figure 27: Configure Fat Flows for test-right-VN


Fat Flows

Protocol: ICMP
Port*: 0
Ignore Address: None

Prefix Aggregation Source
Source Subnet: 198.51.100.0/24
Prefix: 24

Prefix Aggregation Destination
Destination Subnet: 192.0.2.0/24
Prefix: 24

4. Click **Save** to update new configuration information.

   The All Networks page is displayed.

   Repeat steps 2 through 4 to configure fat flows for the **test-right-VN**.

Create Service Policy

Follow these steps to create a service policy by using the Contrail Command UI.

1. Click **Overlay** > **Network Policies**.

   The Network Policies page is displayed.
2. Click **Create**.

   The Network Policy tab of the Create Network Policy page is displayed.
3. Enter **test-network-policy** in the **Policy Name** field.
4. In the **Policy Rule(s)** section,
   - Select **pass** from the **Action** list.
   - Select **ANY** from the **Protocol** list.
   - Select **Network** from the **Source Type** list.
   - Select the **test-left-VN** from the **Source** list.
   - In the **Source Port** field, leave the default option, **Any**, as is.
   - Select **< >** from the **Direction** list.
   - Select **Network** from the **Destination Type** list.
   - Select the **test-right-VN**from the **Destination** list.
   - In the **Destination Ports** field, leave the default option, **Any**, as is.

5. Click **Create** to create the service policy.

   The Network Policies page is displayed. All policies that you created are displayed in the Network Policies page.

Attach Service Policy

Follow these steps to attach a service policy:

1. Click **Overlay**>**Virtual Networks**.

   The All networks page is displayed.
2. Attach service policy to the left virtual network (**test-left-VN**) and right virtual network (**test-right-VN**) that you created.

   To attach service policy,

   a. Select the check box next to the name of the virtual network.

   b. Hover over to the end of the selected row and click the **Edit** icon.

      The Edit Virtual Network page is displayed.

   c. Select the network policy from the Network Policies list.
3. Click **Save** to save the changes.

   The Virtual Networks page is displayed.

Launch Virtual Machine

IN THIS SECTION

You can launch a virtual machine from OpenStack or from Contrail Command UI.

Launch a Virtual Machine from OpenStack

You can launch virtual machines from OpenStack and test the traffic through the service chain by doing the following:

1. Launch the left virtual machine in left virtual network. See "Create Virtual Machines by using OpenStack" on page 127.

2. Launch the right virtual machine in right virtual network. See "Create Virtual Machines by using OpenStack" on page 127.
3. Ping the left virtual machine IP address from the right virtual machine.

   Follow these steps to ping a virtual machine:

   a. Click **Project** > **Compute** > **Instances**.

      All virtual machine instances that you created are displayed on the Instances page.

   b. From the list of virtual machines, click **test-right-VM**.

      The Overview tab of the test-right-VM is displayed.

   c. Click the **Console** tab.

      The Instance Console is displayed.

   d. Log in using the `root` user credentials.

   e. Ping the left virtual machine IP address (**190.0.2.3**) from the Instance Console.

      See Figure 28 on page 137 for a sample output.

Launch a Virtual Machine from Contrail Command

You can launch virtual machines from Contrail Command and test the traffic through the service chain by doing the following:

1. Launch the left virtual machine in left virtual network. See "Create Virtual Machines by using Contrail Command" on page 128.
2. Launch the right virtual machine in right virtual network. See "Create Virtual Machines by using Contrail Command" on page 128.
3. Ping the left virtual machine IP address from the right virtual machine.

   Follow these steps to ping a virtual machine:

   a. Click **Workloads**>**Instances**.

      The Instances page is displayed.

   b. Click the open console icon next to **test-right-VM**.

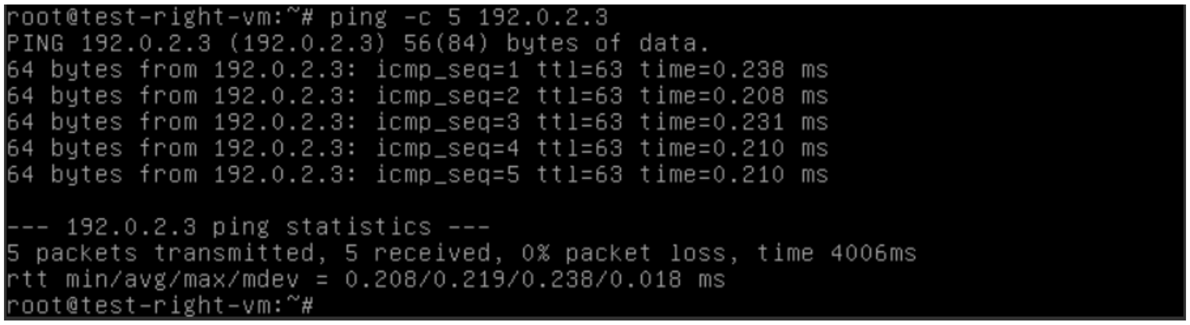
      The Console page is displayed.

   c. Log in using the `root` user credentials.

   d. Ping the left virtual machine IP address (**190.0.2.3**) from the Console.

      See Figure 28 on page 137 for a sample output.

Figure 28: Ping test-left-VM



```
root@test-right-vm:~# ping -c 5 192.0.2.3
PING 192.0.2.3 (192.0.2.3) 56(84) bytes of data.
64 bytes from 192.0.2.3: icmp_seq=1 ttl=63 time=0.238 ms
64 bytes from 192.0.2.3: icmp_seq=2 ttl=63 time=0.208 ms
64 bytes from 192.0.2.3: icmp_seq=3 ttl=63 time=0.231 ms
64 bytes from 192.0.2.3: icmp_seq=4 ttl=63 time=0.210 ms
64 bytes from 192.0.2.3: icmp_seq=5 ttl=63 time=0.210 ms

--- 192.0.2.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 0.208/0.219/0.238/0.018 ms
root@test-right-vm:~#
```

RELATED DOCUMENTATION

Fat Flows | 105

Understanding Flow Sampling

IN THIS SECTION

This topic describes how flow records are sampled and exported to the Contrail collector, flow handling, and flow aging.

Flow Sampling

The Contrail vRouter agent exports flow records to the Contrail collector when a flow is created or deleted. It also updates flow statistics at regular intervals.

If all flow records are exported from the agent, depending on the scale of flows, some of the exported flows might be dropped due to queue overflow.

Based on sampling, flow records are sampled and exported to the Contrail Controller of Contrail Networking. This enables Contrail Networking to reduce queue overflow.

The flows that are exported are selected based on the following parameters used in the algorithm:

- The configured flow export rate. This is configured as part of the `global-vrouter-config` object.
- The actual flow export rate.
- The sampling threshold. This is a dynamic value calculated internally. If the flow statistics in a flow sample are above this threshold, the flow record is exported.

Each flow is subjected to the following algorithm at regular intervals. The algorithm determines whether to export the sample or not.

- Flows with traffic that is greater than or equal to the sampling threshold are always exported. The byte and packet counts are reported without modification.
- Flows with traffic that is less than the sampling threshold are exported according to the probability. The byte and packet counts are adjusted upwards according to the probability.

  The probability is calculated as (bytes during the interval) / (sampling threshold).
- The system generates a random number less than the sampling threshold. If the byte count during the interval is less than the random number, then the flow sample is not exported.
- If none of these conditions are met, the flow sample is exported after normalizing the byte count and packet count during the interval. Normalization is done by dividing the byte count and packet count during the interval by the probability. This normalization is used as a heuristic to account for statistics of flow samples that are dropped.

The actual flow export rate is close to the configured export rate. If there is a large deviation, the sampling threshold is adjusted to bring the actual flow export rate close to the configured flow export rate.

Flow Handling

When a virtual machine sends or receives IP traffic, forward and reverse flow entries are set up. When the first packet arrives, a flow key is used to hash into a flow table (identify a hash bucket). The flow key is based on five-tuples consisting of source and destination IP addresses, ports, and the IP protocol.

A flow entry is created and the packet is sent to the Contrail vRouter agent. Configured policies are applied and the flow action is updated. The agent also creates a flow entry for the reverse direction where relevant. Subsequent packets match the established flow entries and are forwarded, dropped, NAT translated, and so on, based on the flow action.

When the hash bucket is full, entries are created in an overflow table. Contrail Networking maintains the overflow entries as a list against the hash bucket.

By default, the maximum number of flow table and overflow table entries are 512,000 and 8000 respectively. These can be modified by configuring them as vRouter module parameters, for example: `vr_flow_entries` and `vr_oflow_entries`.

For more information about the vRouter module parameters, see https://github.com/Juniper/contrail-controller/wiki/Vrouter-Module-Parameters.

Flow Aging

Flows are aged out based on inactivity for a specified period of time. By default, the timeout value is 180 seconds. This can be modified by configuring the `flow_cache_timeout` parameter under the `DEFAULT` section in the `/etc/contrail/contrail-vrouter-agent.conf` file.

TCP State-Based Flow Handling and Aging

TCP State-Based Flow Handling

The Contrail vRouter in Contrail Networking monitors TCP flows to identify entries that can be reused without going through the standard aging cycle.

All flow entries that match TCP flows that have experienced a connection teardown, either through the standard TCP closure cycle (FIN/ACK-FIN/ACK) or the RST indicator, are torn down by the vRouter and are immediately available for use by new qualified flows.

The vRouter also keeps track of connection establishment cycles and exports the necessary information to the vRouter agent, such as SYN/ACK and a digested established flag. This allows the vRouter agent to tear down flows that do not experience a full connection cycle.

Flows that the vRouter identifies as reuse candidates, or eviction candidates, are marked as such in the flow entry. Flows are in the evicted state when they become available for other new flows to be reused.

This two-step transition is used so that the flow entry remains valid until the packet reaches the destination, preventing the flow from getting remapped to another flow entry in the interim.

Protocol-Based Flow Aging

Although TCP flows are deleted based on TCP state, you are sometimes required to age out specific protocol flows more aggressively. One example is when a DNS server is run in one VM. In this case, multiple flows are set up for DNS. A pair of flows are set up to serve each query. Because the flows are no longer required after the query is served, the timeout can be lower for these flows. To handle these cases, protocol-based flow aging is used.

With protocol-based flow aging, the aging timeout can be configured per protocol. All other protocols continue to use the default aging timeout.

Protocol-based flow aging is also supported in Contrail Networking.

The configuration for protocol-based flow aging can be done in the `global-vrouter-config` object. For example, to have all DNS flows aged out in five seconds, use the following entry: `protocol = udp, port = 53 will be set an aging timeout of 5 seconds.`

4

CHAPTER

OpenStack Features Support in Contrail

Neutron Port Binding API

OpenStack Neutron uses new port binding API to assist with live migration scenarios and has deprecated the old methodology that impacts live migration of a virtual machine to work in RHOSP 17. The main motivation for implementing this feature is selection of problems around live-migration like port binding failure at destination after the live migration starts. With the implementation of this feature, Contrail can detect whether port binding will be successful at destination before starting live migration.

Earlier, live migration uses single port binding. Now, OpenStack live migration uses neutron port binding API to model both source and destination hosts having port binding, which is used during live migration.

Contrail Neutron plugin is updated to receive, process and send appropriate responses from Nova for live migration using new Neutron port binding API. Nova live migration consists of the following stages:

- Pre-live-migration
- Live-migration-operation
- Post-live-migration

For more information, refer to OpenStack live migration document: Live-migration — Neutron 24.0.0.0b2.dev116 documentation.